EXHIBIT 2

U.S. Patent No. 9,615,192 ("'192 Patent")

Accused Devices: Samsung's push messaging servers (e.g., Samsung's Knox and Tizen servers), and all versions and variations thereof since the issuance of the asserted patent.

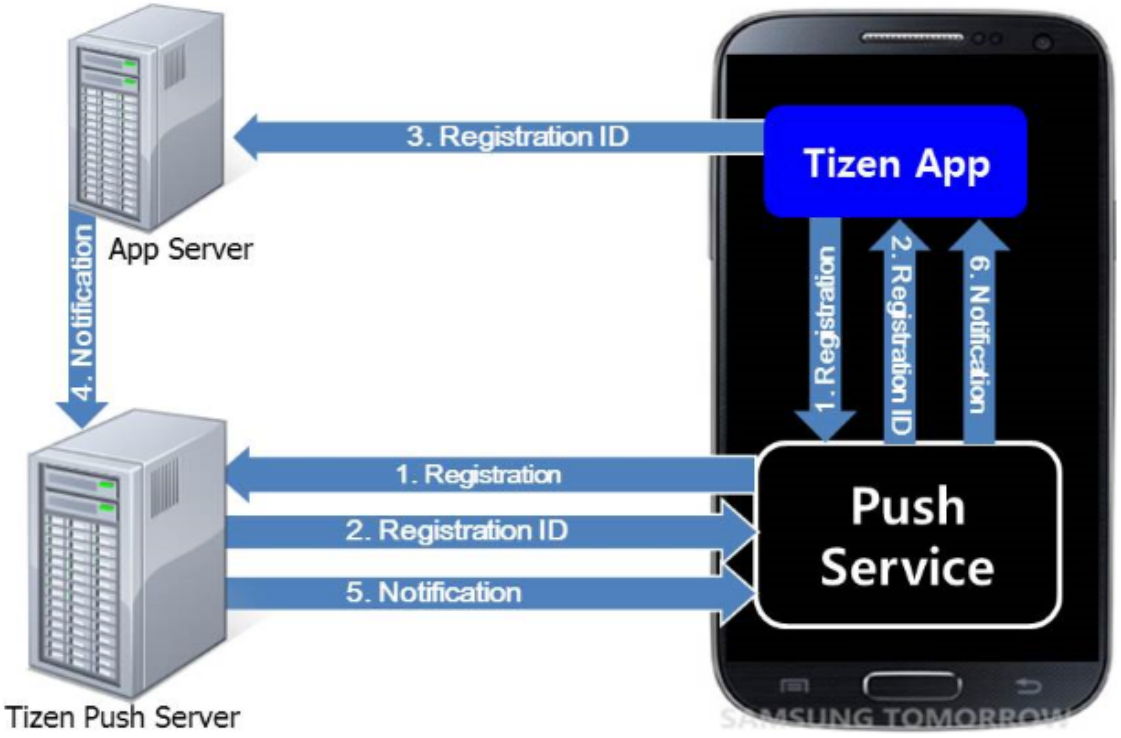
Claim 1

Issued Claim(s)	Public Documentation
[1pre]. A message link server comprising:	Samsung's push messaging servers are each "message link server[s]." *See, e.g.,*

Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

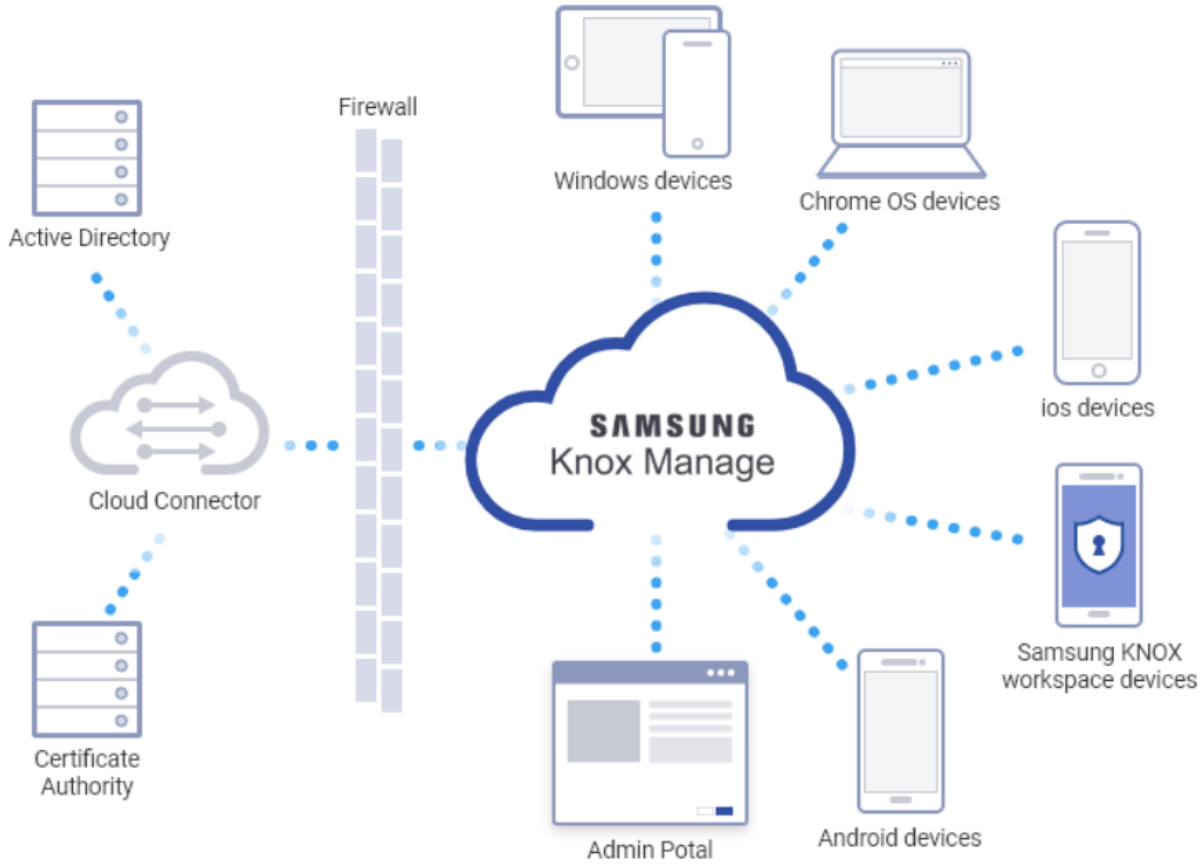
Figure: Service architecture



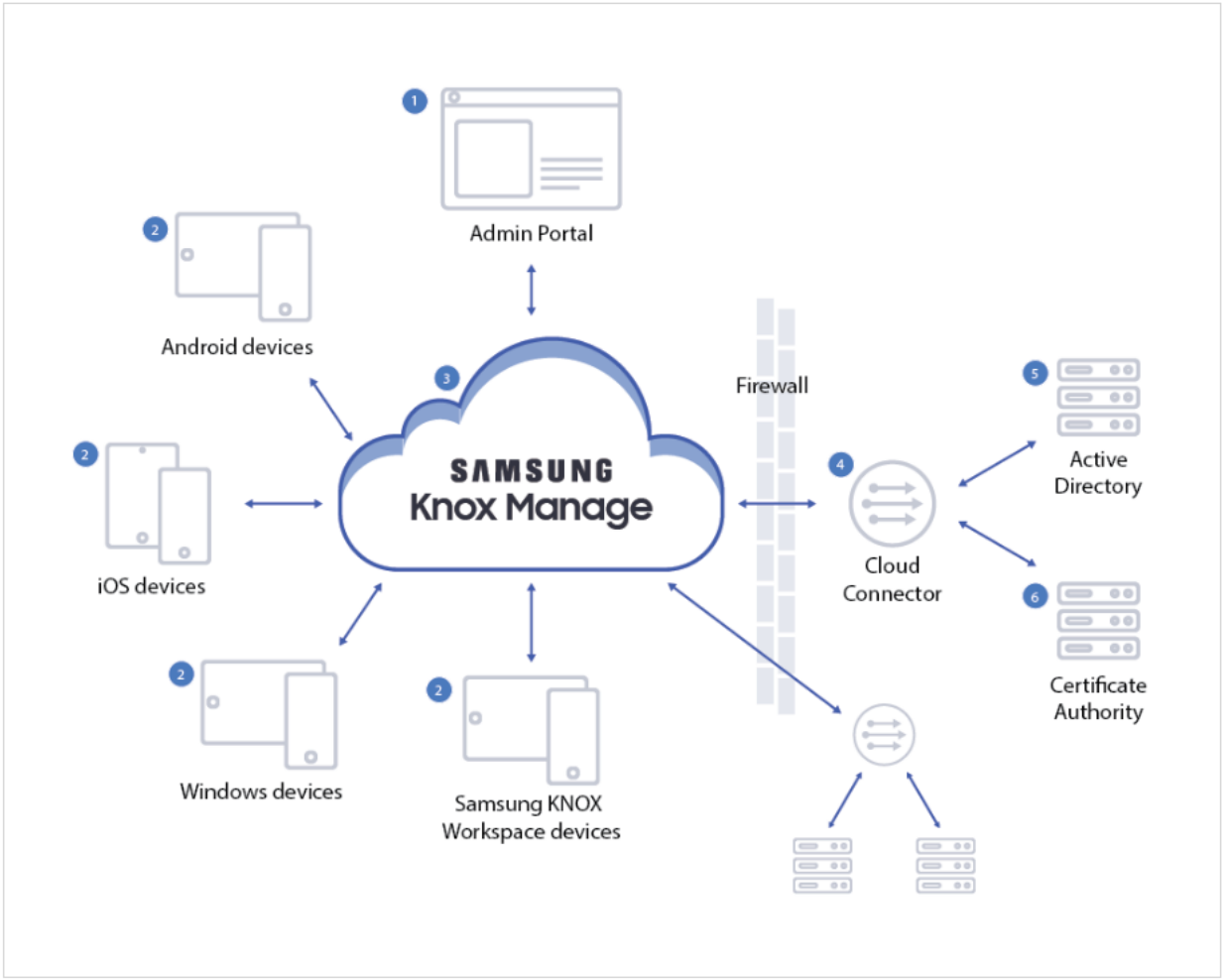
To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

https://docs.tizen.org/application/web/guides/messaging/push/;



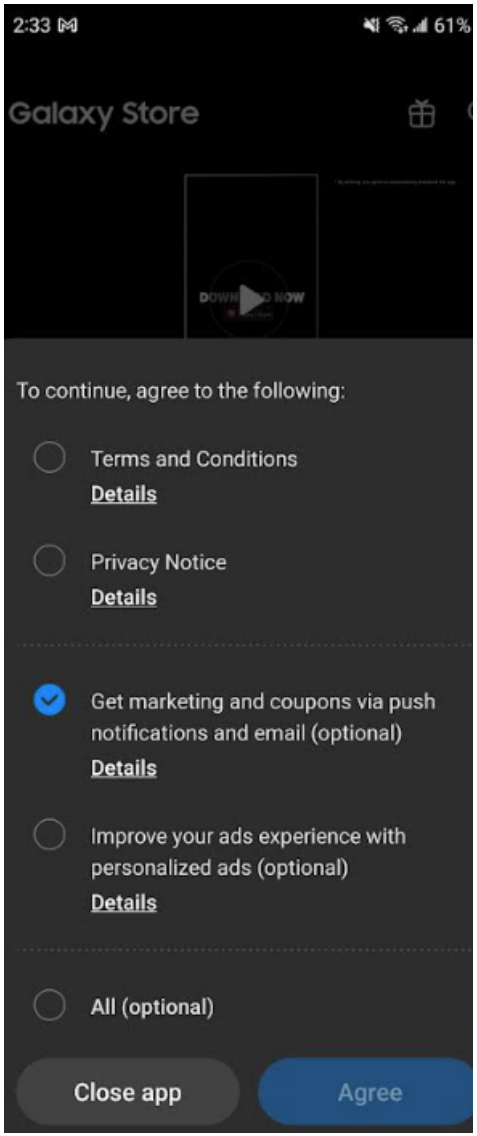
https://docs.samsungknox.com/admin/knox-manage/welcome.htm



https://docs.samsungknox.com/admin/knox-manage/km-features.htm

On information and belief, Samsung owns and operates its own push messaging servers, including servers for communicating with the Samsung Push Service software on its end user devices. *See, e.g.*, https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1; https://www.lifewire.com/samsung-push-service-4165507. On information and belief, Samsung includes applications in its user devices, such as the "Shop Samsung" and "Galaxy Store"

applications, which cause notifications (including marketing notifications) to be sent to user devices via Samsung's push messaging servers. *See, e.g.*:

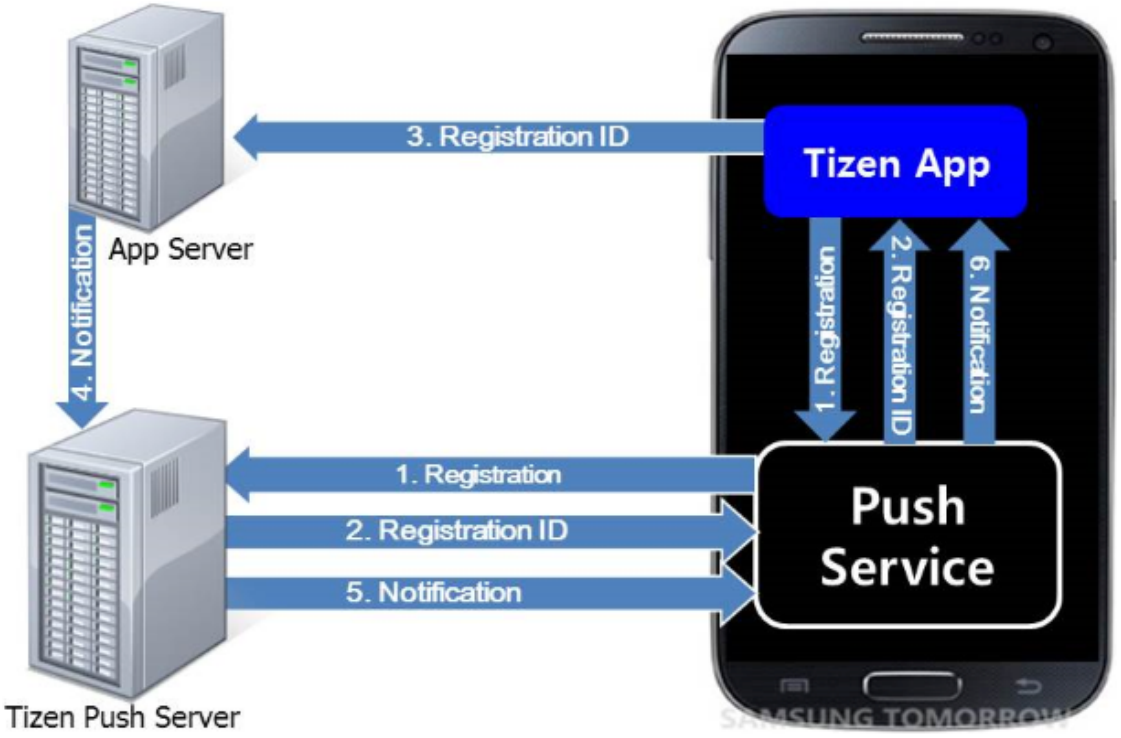


[1a] a transport services stack to maintain a respective secure message link through an Internet network between the message link server and a respective device link agent on each of a plurality of wireless end-user devices, each of the wireless end-user devices comprising multiple software components authorized to receive and process data from secure message link messages received via a device link agent on that device;	Samsung's push messaging servers comprise "a transport services stack to maintain a respective secure message link through an Internet network between the message link server and a respective device link agent on each of a plurality of wireless end-user devices, each of the wireless end-user devices comprising multiple software components authorized to receive and process data from secure message link messages received via a device link agent on that device." For example, Samsung's push messaging server maintains a secure message link between the Tizen Push Server and a device link agent (e.g., software in the Accused Products responsible for receiving, decrypting, delivering, and displaying messages received by the Accused Products, such as the Tizen App and/or Push Service) on a plurality of devices which comprise multiple software components (e.g., operating system software and/or applications) and receive/process data from messages sent over the secure message link using a transport stack (e.g., the software and configurations that allow connections to be established and secured and which are used to transmit data from the server to the device). *See, e.g.,* **Push Notification** You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests. The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators. Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer. If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application.

Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

- Up to Tizen 2.4:

  1. Define event handlers for the registration results:

```
/*
   Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

  2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```

- Since Tizen 3.0:

  Before registering, you must connect to the push service:

  1. Define event handlers:

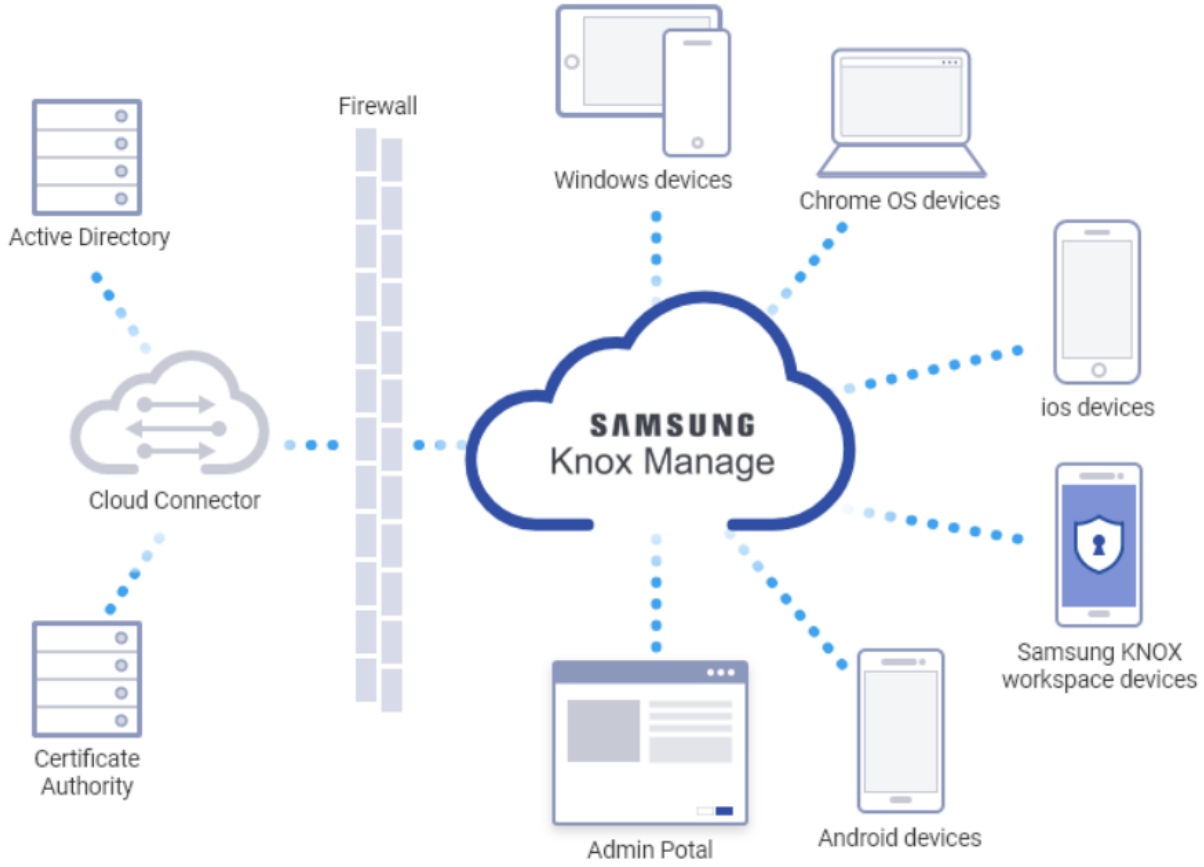
```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);

    if (state == 'UNREGISTERED') {
        /* Request application registration */
        tizen.push.register(registerSuccessCallback, errorCallback);
    }
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
}
```

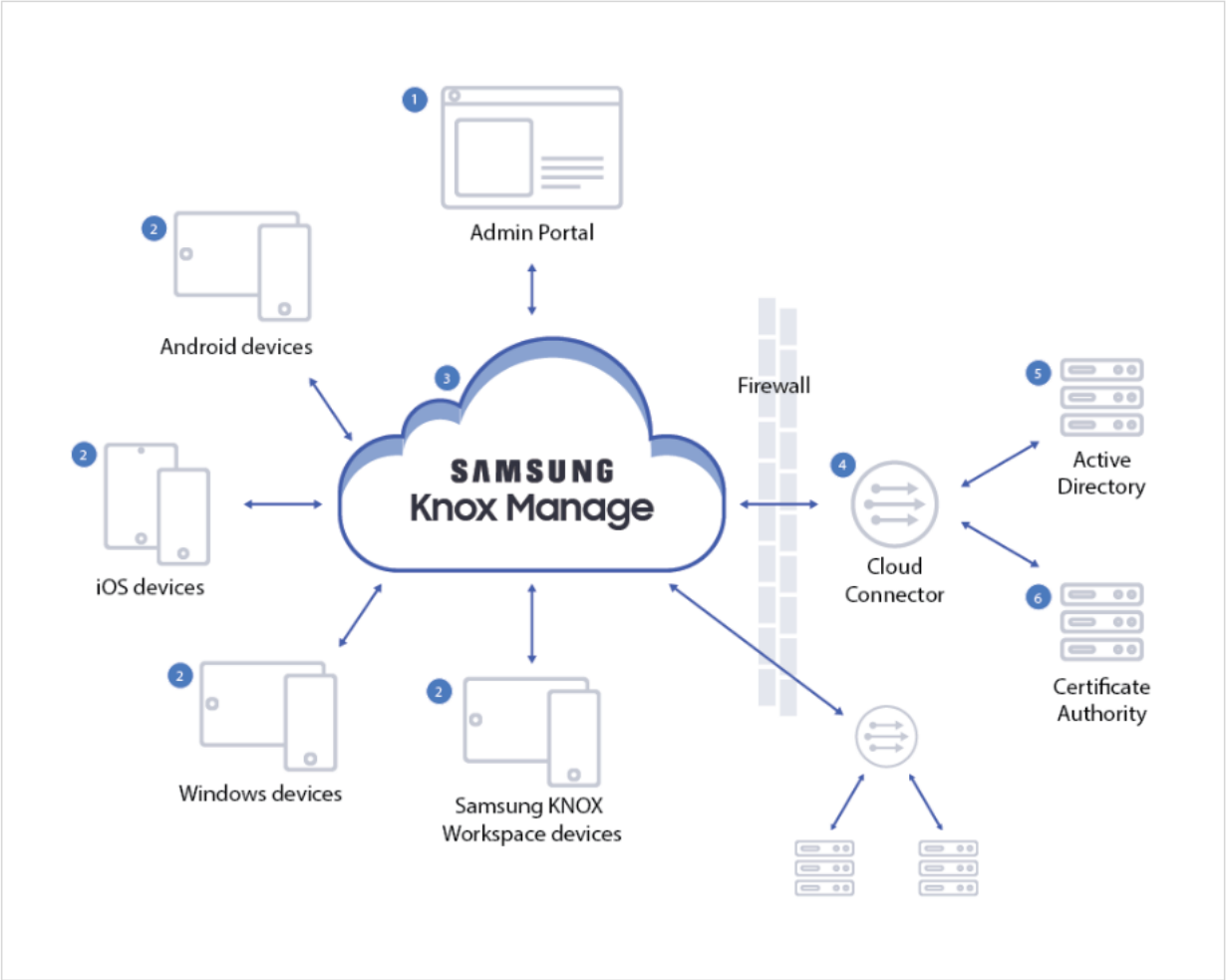
  2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```

https://docs.tizen.org/application/web/guides/messaging/push/.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, maintains a secure message link between the server and a device link agent (e.g., software in the Accused Products responsible for receiving, decrypting, delivering, and displaying messages received by the Accused

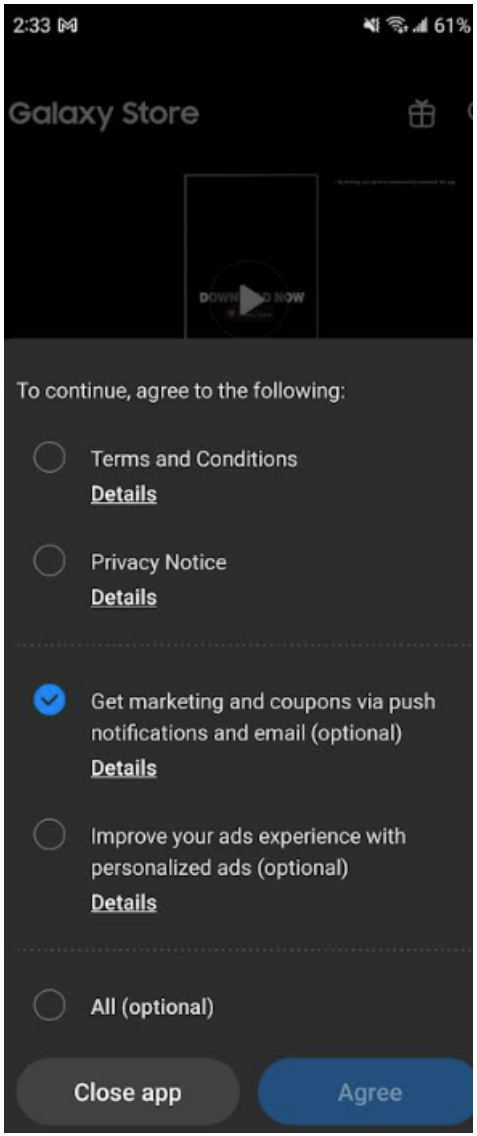
	Products, such as the Knox app and/or other software applications) on a plurality of devices which comprise multiple software components and receive/process data from messages sent over the secure message link using a transport stack (e.g., the software and configurations that allow connections to be established and secured and which are used to transmit data from the server to the device)..  https://docs.samsungknox.com/admin/knox-manage/welcome.htm



https://docs.samsungknox.com/admin/knox-manage/km-features.htm

On information and belief, Samsung owns and operates its own push messaging servers, including servers for communicating with the Samsung Push Service software on its end user devices. *See, e.g.*, https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1; https://www.lifewire.com/samsung-push-service-4165507. On information and belief, Samsung includes applications in its user devices, such as the "Shop Samsung" and "Galaxy Store"

applications, which cause notifications (including marketing notifications) to be sent to user devices via Samsung's push messaging servers. *See, e.g.*:

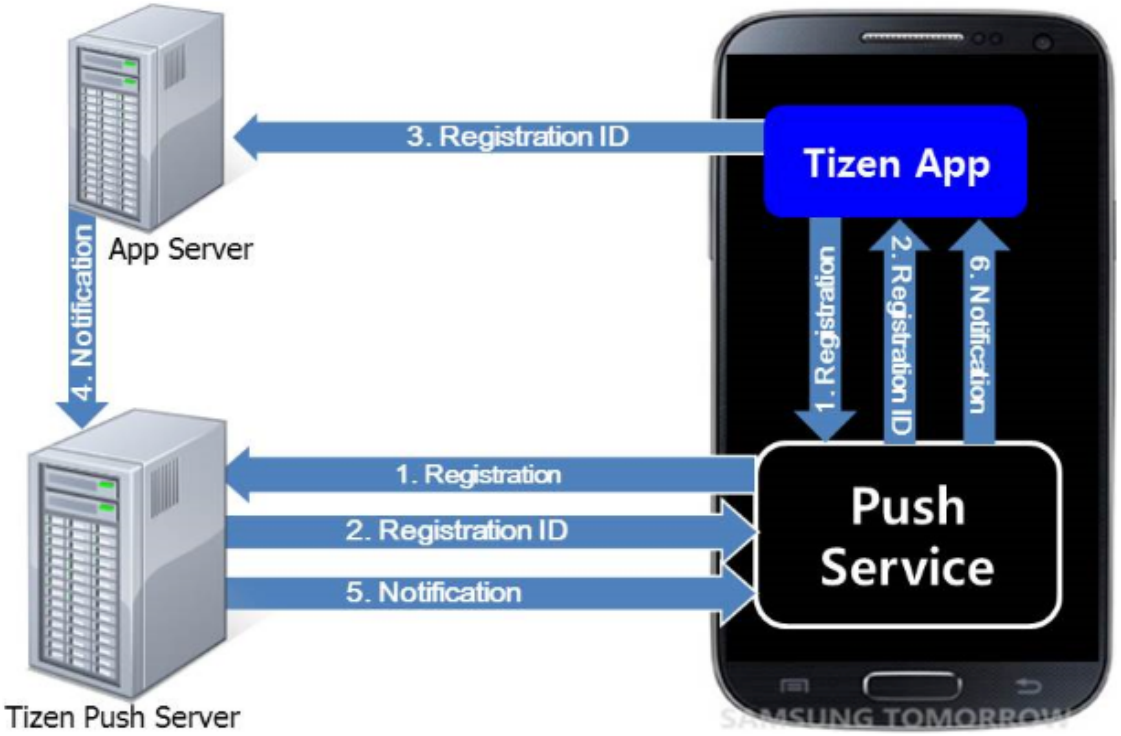


	As a further example, the communications link Samsung Knox Manage servers or Knox MDM platform and devices being managed by such servers or MDM platform, for example including the Knox workspace devices, Android devices, iOS devices, Chrome OS devices, and Windows devices illustrated above, comprise a "service control link." Data and messages transmitted over the secure message link between the Samsung Knox Manage servers or Knox MDM platform and managed devices are encrypted, and commands and notifications transmitted from such servers to devices are messages. As a further example, the communications link between Samsung's Tizen servers and Tizen OS devices, and Firebase messaging servers and Samsung's Android devices, comprises a "service message link." Notifications, data, and messages transmitted over these service control links from such servers to devices are encrypted, as illustrated above, and are messages.
[1b] an interface to a network to receive network element messages from a plurality of network elements, the received network element messages comprising respective message content and requests for delivery of the respective message content to respective wireless end-user devices, the respective message content including data for, and an identification of, a respective one of the authorized software components; and	Samsung's push messaging servers comprise "an interface to a network to receive network element messages from a plurality of network elements, the received network element messages comprising respective message content and requests for delivery of the respective message content to respective wireless end-user devices, the respective message content including data for, and an identification of, a respective one of the authorized software components." For example, Samsung's push messaging servers receive messages from a plurality of network elements (e.g., App Servers) which comprise content and delivery requests to devices and software components. *See, e.g.,*

Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

https://docs.tizen.org/application/web/guides/messaging/push/;

Push Server

You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the error code to determine how to handle the failure.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

The main features of the Push API for the server developers include:

- Sending push notifications

  You can send push notifications from the application server to an application.

- Decorating push notifications

  You can add decorations to the push notifications in the quick panel.

https://docs.tizen.org/application/native/guides/messaging/push-server/;

Push

You can push events from an application server to your application on a Tizen device.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

Once your application is successfully registered in the push server through the push service (daemon) on the device, your application server can send push messages to the application on that particular device.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application as an application control.

Figure: Push messaging service

Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the Push Server guide for server developers.

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

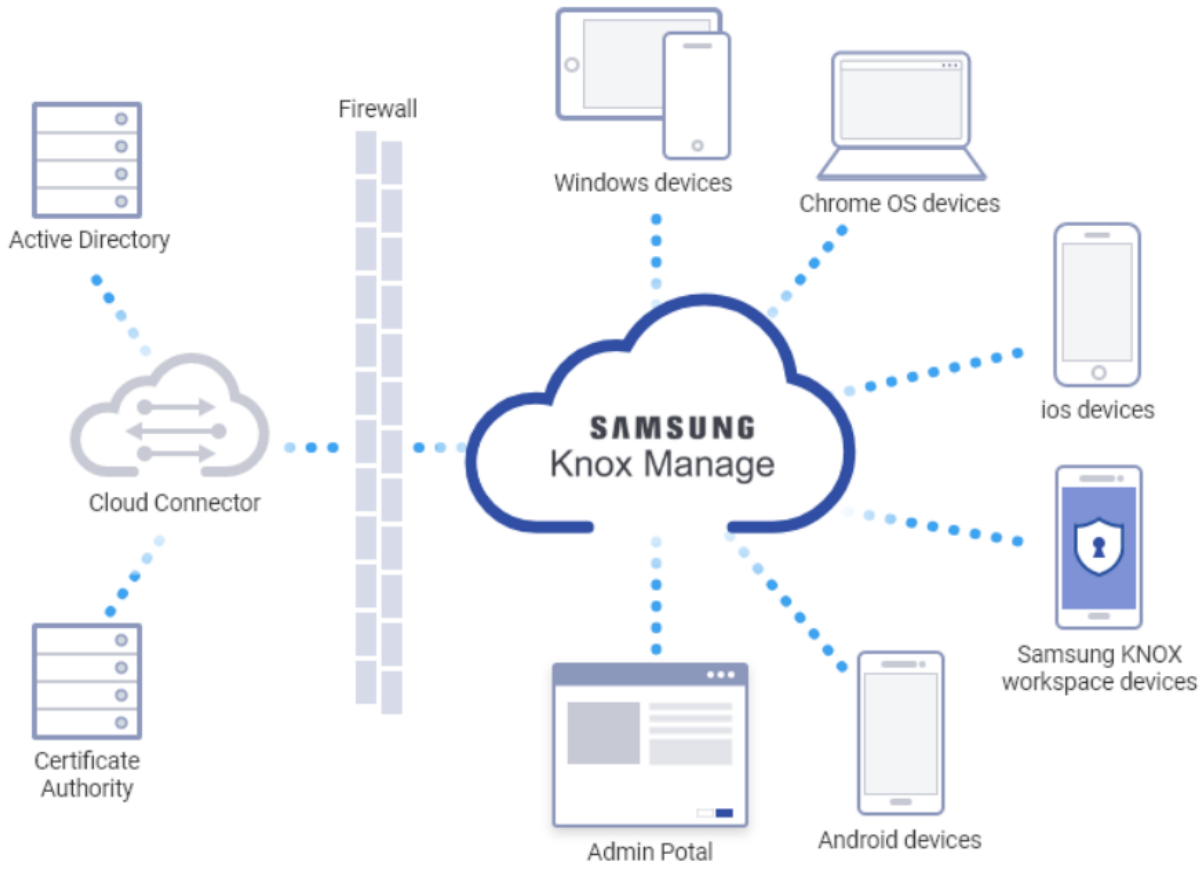
```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

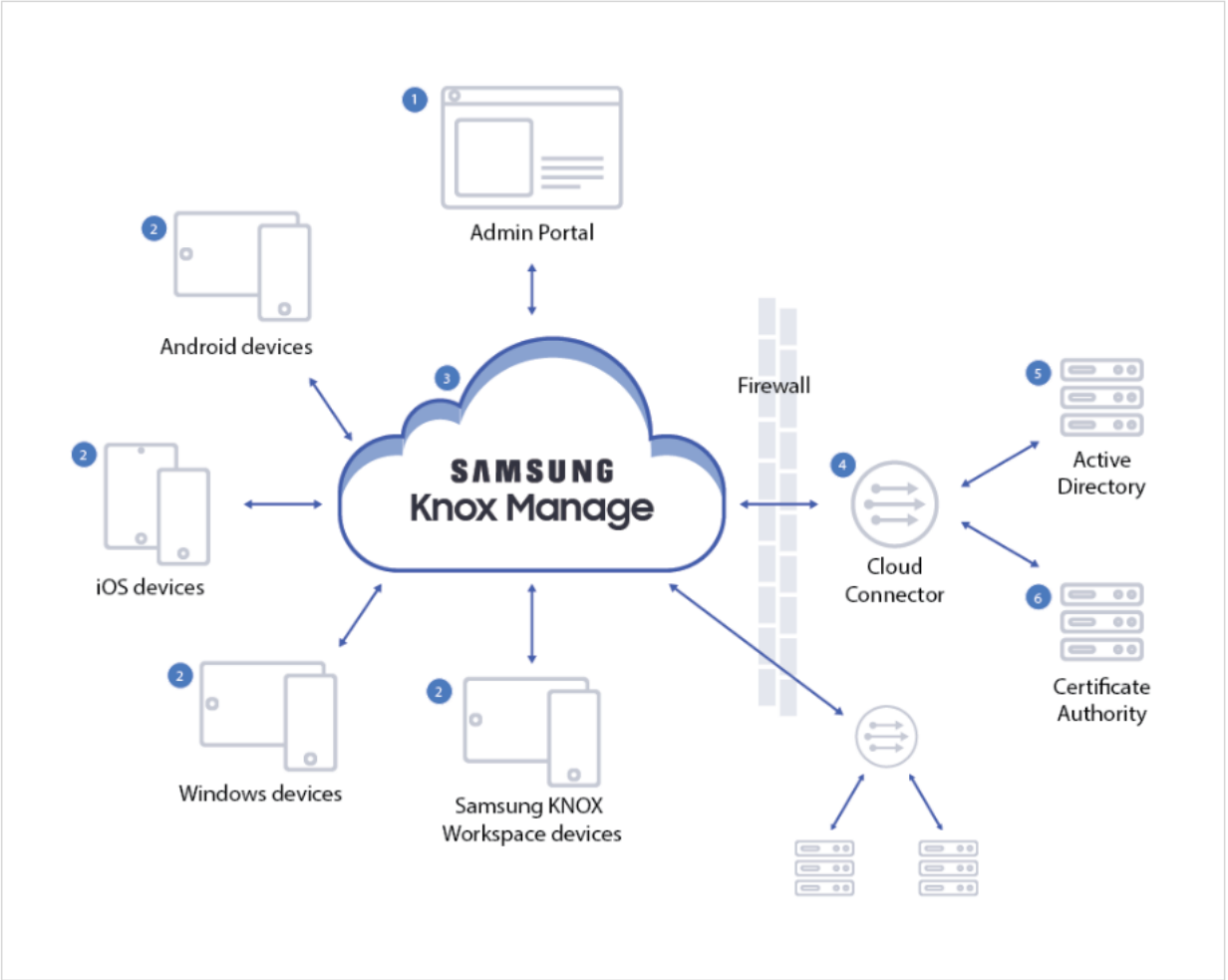
To send a notification:

1. Prepare the `appID`, `appSecret`, `regID`, and `requestID`:
    - The `appID` and `appSecret` values are given in the email message that you received when requesting permission to use Tizen push servers.
    - The `regID` value is the one that the application server received from your application installed on a Tizen device. Depending on the `regID` value, the URI of the server to which your application server sends the notification varies.
    - The `requestID` value is used to identify the notification in the push server. When your application server sends notifications using the same `requestID` value, the last notification overwrites all the previous notifications that are not delivered yet.
2. Use the message field to describe how to process the notification.

   The message field contains not only the message to show in the quick panel on the device, but also the behaviors that the device must take when receiving the notification. The message field is a string that consists of key-value pairs. The available pair options are given in the following table.

https://docs.tizen.org/application/native/guides/messaging/push/.

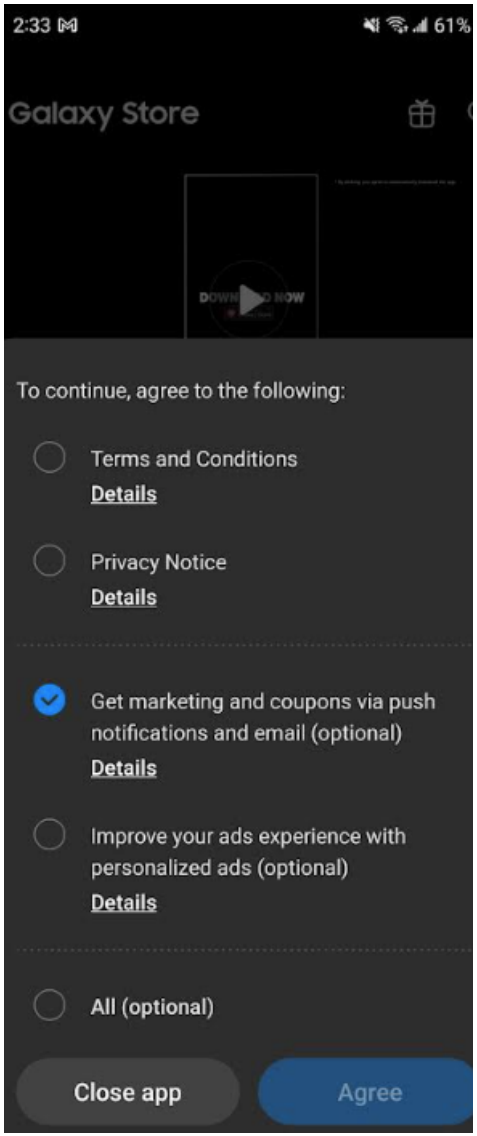
	As another example, Samsung's receive messages from a plurality of network elements (e.g., App Servers) which comprise content and delivery requests to devices and software components in the Knox ecosystem.  https://docs.samsungknox.com/admin/knox-manage/welcome.htm



https://docs.samsungknox.com/admin/knox-manage/km-features.htm

On information and belief, Samsung owns and operates its own push messaging servers, including servers for communicating with the Samsung Push Service software on its end user devices. *See, e.g.*, https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1; https://www.lifewire.com/samsung-push-service-4165507. On information and belief, Samsung includes applications in its user devices, such as the "Shop Samsung" and "Galaxy Store"

applications, which cause notifications (including marketing notifications) to be sent to user devices via Samsung's push messaging servers. *See, e.g.*:



	As a further example, the App Servers communicating with Tizen and Firebase notification servers (each a network element) send messages such as notifications (network element messages comprising respective message content and request for delivery of the respective message content), which are routed through Samsung's Tizen and Knox servers and Google's Firebase messaging servers. Alternatively, administrator devices communicating with Knox MDM and Knox Manage servers to manage and control devices in the Knox device group are also network elements which send to the Knox servers messages that comprise message content and requests for delivery of the message content down to the managed devices. Such messages can be device commands transmitted from the administrator devices to managed devices via Knox servers (e.g., https://docs.samsungknox.com/admin/knox-manage/send-commands-to-devices.htm) or readable notifications. Alternatively, each of the managed devices in a Knox MDM group can also comprise network elements sending messages and requests for delivery of message content, for example when agents on the managed device monitor and trigger alerts to be sent to administrator devices via Knox MDM servers (e.g., https://docs.samsungknox.com/admin/knox-manage/configure-alerts.htm). In each example, such messages include data for, and identification of, a respective one of the authorized software components (e.g., the application to which the notification and message is delivered, whether it is the Knox Agent on the devices, the Knox Manage application on the administrator device, or the particular application on the Tizen device receiving a notification).
[1c] a message buffer system including a memory and logic,	The Accused Devices include "a message buffer system including a memory and logic." For example, Samsung's push messaging servers include memory (e.g., storage or RAM) to store messages (e.g., a buffer system to store messages) and logic (e.g., logic to determine when, how, and under what circumstances to store and/or deliver messages). *See e.g.*,

When a notification arrives at the device, its delivery mechanism depends on whether the application is running:

- When the application is running

  When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case.

- When the application is not running

  If the notification arrives when the application is not running, there are 3 ways to handle the notification:

  - Forcibly launch the application and deliver the notification to it.

    This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

    For more information, see the Retrieving Messages When Launched by the Push Service use case.

  - Store the notification at the push service database and request it later when the application is launched.

    This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

    For more information, see the Retrieving Missed Push Messages use case.

    The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

  - Discard the notification.

    This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

https://docs.tizen.org/application/web/guides/messaging/push/;

If the notification arrives when the application is not running, it can be handled in 3 ways:

- Forcibly launch the application and deliver the notification to it.

  You need to set the action to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

  When you create a project in the Tizen Studio, the `app_control()` function is created automatically. When the application is launched by another application or process (in this case, by the push service), all related information regarding this launch request is delivered through the `app_control` parameter. From this handle, retrieve the `op` operation using the `app_control_get_operation()` function. With `app_control` and `op`, retrieve the notification data using the `push_service_app_control_to_noti_data()` function.

  If the application is not launched by the push service, this function returns as `NULL`.

```
static void
app_control(app_control_h app_control, void *data)
{
    char *op = NULL;
    push_service_notification_h noti = NULL;
    int ret;

    if (app_control_get_operation(app_control, &op) < 0)
        return;

    /* Retrieve the noti from the bundle */
    ret = push_service_app_control_to_notification(app_control, op, &noti);

    if (noti) {
        /* Handle the noti */

        /* Free the noti */
        push_service_free_notification(noti);
    } else {
        /* Case when the application is not launched by the push service */
    }
    if (op)
        free(op);
}
```

Since Tizen 3.0, the push service provides launch types when the application is launched by the service. Use the following code to figure out why the application is launched, as the `app_control()` function is invoked in both cases of receiving notification and changing registration state.

```
#define EXTRA_DATA_FROM_REGISTRATION_CHANGE "registration_change"
#define EXTRA_DATA_FROM_NOTIFICATION "notification"

static void
app_control(app_control_h app_control, void *data)
{
    char *value = NULL;
    app_control_get_extra_data(app_control, APP_CONTROL_DATA_PUSH_LAUNCH_TYPE, &value);
    if (value) {
        if (!strcmp(value, EXTRA_DATA_FROM_NOTIFICATION))
            /* Add your code here when push messages arrive */
        else if (!strcmp(value, EXTRA_DATA_FROM_REGISTRATION_CHANGE))
            /* Add your code here when registration state is changed */
    }
}
```

- Store the notification at the push service database and request it later when the application is launched.

  You need to set the action to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

  You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched.

```
if (push_conn) {
    int ret = push_service_request_unread_notification(push_conn);
    if (ret != PUSH_SERVICE_ERROR_NONE)
        dlog_print(DLOG_ERROR, LOG_TAG, "ERROR: push_service_request_unread_notification()
failed.");
}
```

  The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification through the app control callback function.

- Discard the notification.

  You need to set the action to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service delivers the notification only when the application is up and running. Otherwise, the push service does not store the notification and discards it.

- Request unread notifications.

  If the user does not launch the application from the quick panel, the application requests the unread notifications after start-up using the asynchronous `push_service_request_unread_notification()` function.

  The following example shows a synchronous request using the `push_service_get_unread_notification()` function:

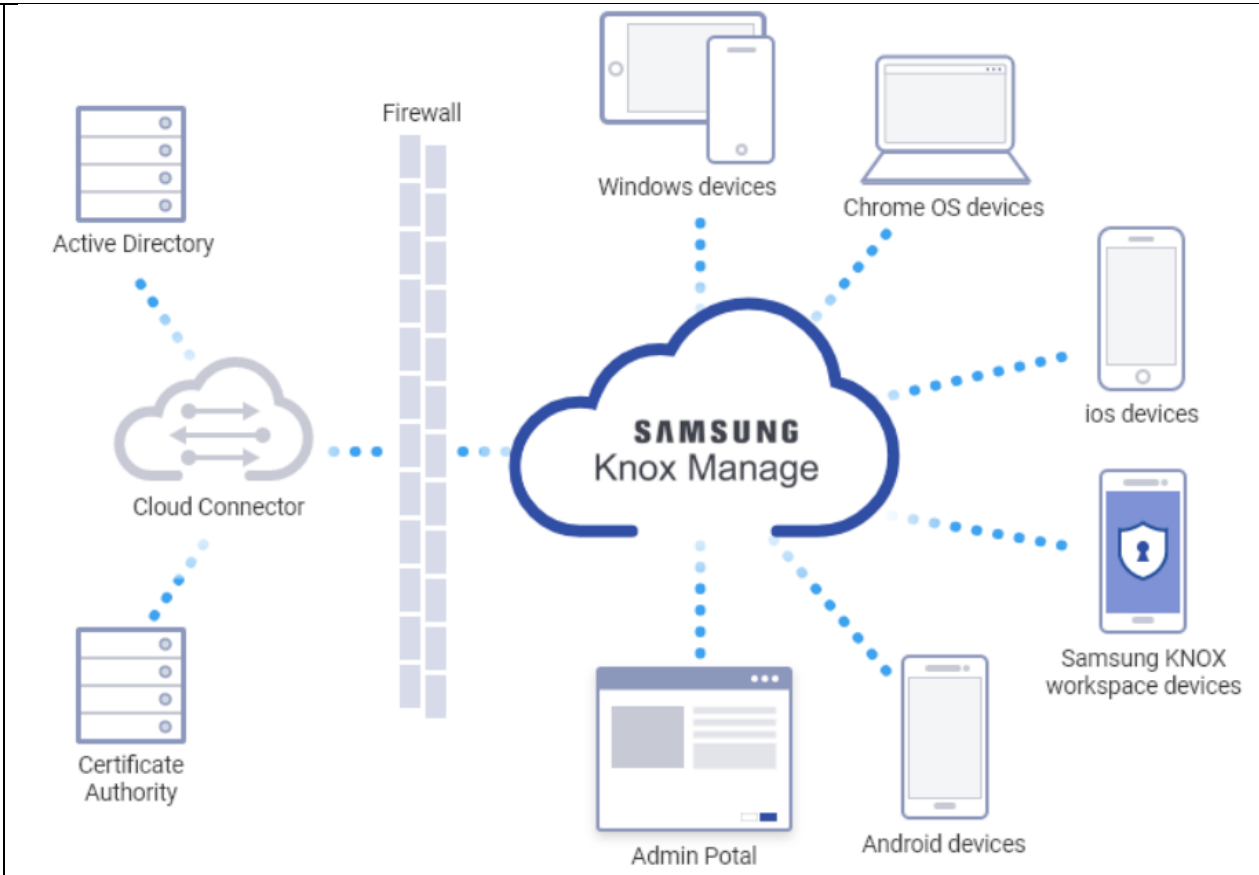
```
push_service_notification_h noti;
int ret;
do {
    ret = push_service_get_unread_notification(push_conn, &noti);

    /* Process the unread message noti */

    push_server_free_notification(&noti);
} while (1);
```

  Call this function repeatedly until no notification is returned. If there are multiple unread notifications, the notifications are retrieved in their arrival order.

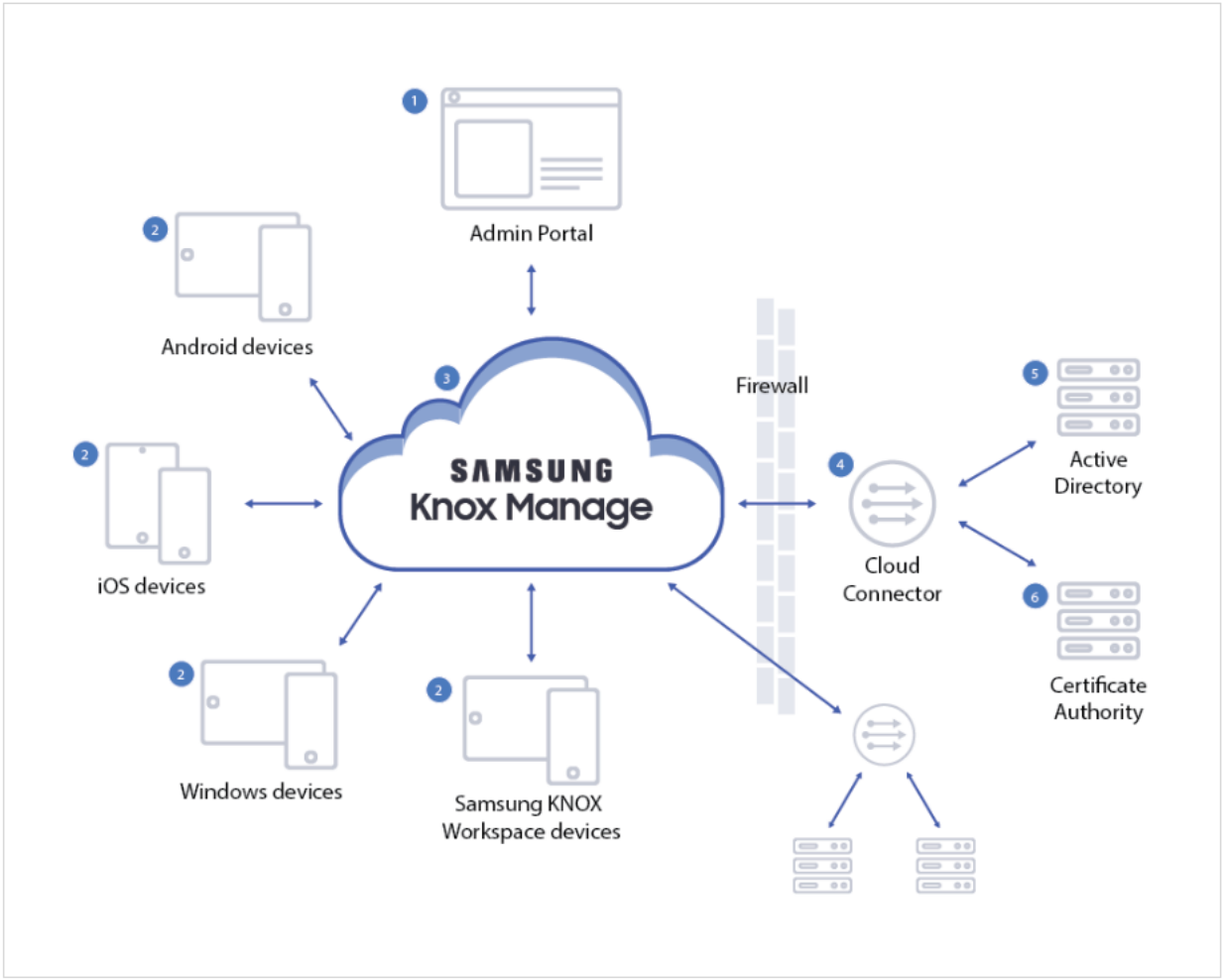
  The `push_service_get_unread_notification()` function blocks the code while it receives a notification from the service. Unless you need this kind of synchronous behavior, use the asynchronous function.

https://docs.tizen.org/application/native/guides/messaging/push/.

As another example, Samsung's Knox servers include memory (e.g., storage or RAM) to store messages (e.g., a buffer system to store messages) and logic (e.g., computer hardware, software, and instructions to determine when, how, and under what circumstances to store and/or deliver messages).



https://docs.samsungknox.com/admin/knox-manage/welcome.htm



https://docs.samsungknox.com/admin/knox-manage/km-features.htm

Update an existing device profile

22.11 23.03 UAT

An admin can update the device's profile with a push update if a device is currently in a Configured state.

> **NOTE** — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile.

> **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not.

6. If displayed, select the **Push update without requesting end user consent** to push the updates directly to the configured state devices without consent from the device end user. Leave this option unselected affords the device user the ability to approve the profile update before its pushed to their device.
7. Select **YES** to proceed with the device push update and profile overwrite. Select **Back** to move back to the previous **Assign devices with profiles** screen.

https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm

	Components of Knox Manage 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm

Set the profile update schedule

You can set the schedule to update the latest policy applied to user devices on a regular basis.

You can define multiple schedules. Click + to add a schedule.

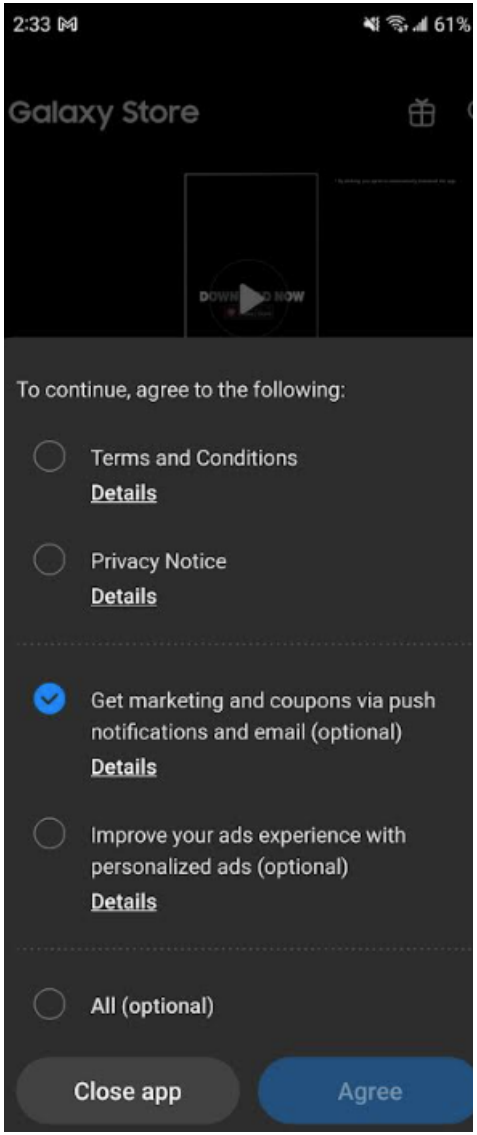
You can add or edit up to 20 schedules when you save the settings.

To configure a policy update schedule:

1. Navigate to **Setting** > **Configuration** > **Profile Update Scheduler**.
2. Click next to Profile Update Schedule to enable the scheduler feature.
3. Select a target type from the following.
   - **Global Setting**—All profiles are updated according to the schedule.
   - **Set by Group / Organization**—You can configure multiple schedules and select groups or organizations for each schedule setting.
4. Select days and set the start time and time zone.
   - Select groups or organizations for each schedule setting if you selected the group/organization target type.
   - The policy update time may vary depending on the time and time zone of the user device.
5. Click **Save & Apply**.

https://docs.samsungknox.com/admin/knox-manage/set-the-profile-update-schedule.htm.

On information and belief, Samsung owns and operates its own push messaging servers, including servers for communicating with the Samsung Push Service software on its end user devices. *See, e.g.*, https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1; https://www.lifewire.com/samsung-push-service-4165507. On information and belief, Samsung includes applications in its user devices, such as the "Shop Samsung" and "Galaxy Store" applications, which cause notifications (including marketing notifications) to be sent to user devices via Samsung's push messaging servers. *See, e.g.*:


2:33
61%
Galaxy Store
To continue, agree to the following:
Terms and Conditions
Details
Privacy Notice
Details
Get marketing and coupons via push notifications and email (optional)
Details
Improve your ads experience with personalized ads (optional)
Details
All (optional)
Close app
Agree

[1d] the memory to buffer content from the received network element messages for which delivery is requested to a given one of the wireless end-user devices,	The Accused Devices include "the memory to buffer content from the received network element messages for which delivery is requested to a given one of the wireless end-user devices." For example, Samsung's push messaging servers include memory (e.g., storage or RAM) to store content (e.g., a buffer system to store data) from received messages for delivery to a device, such as by storing a message until the application is running. *See e.g.*, When a notification arrives at the device, its delivery mechanism depends on whether the application is running: • When the application is running When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case. • When the application is not running If the notification arrives when the application is not running, there are 3 ways to handle the notification: ◦ Forcibly launch the application and deliver the notification to it. This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle. For more information, see the Retrieving Messages When Launched by the Push Service use case. ◦ Store the notification at the push service database and request it later when the application is launched. This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. For more information, see the Retrieving Missed Push Messages use case. The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification. ◦ Discard the notification. This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running. https://docs.tizen.org/application/web/guides/messaging/push/;

If the notification arrives when the application is not running, it can be handled in 3 ways:

- Forcibly launch the application and deliver the notification to it.

  You need to set the action to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

  When you create a project in the Tizen Studio, the `app_control()` function is created automatically. When the application is launched by another application or process (in this case, by the push service), all related information regarding this launch request is delivered through the `app_control` parameter. From this handle, retrieve the `op` operation using the `app_control_get_operation()` function. With `app_control` and `op`, retrieve the notification data using the `push_service_app_control_to_noti_data()` function.

  If the application is not launched by the push service, this function returns as `NULL`.

```
static void
app_control(app_control_h app_control, void *data)
{
    char *op = NULL;
    push_service_notification_h noti = NULL;
    int ret;

    if (app_control_get_operation(app_control, &op) < 0)
        return;

    /* Retrieve the noti from the bundle */
    ret = push_service_app_control_to_notification(app_control, op, &noti);

    if (noti) {
        /* Handle the noti */

        /* Free the noti */
        push_service_free_notification(noti);
    } else {
        /* Case when the application is not launched by the push service */
    }
    if (op)
        free(op);
}
```

Since Tizen 3.0, the push service provides launch types when the application is launched by the service. Use the following code to figure out why the application is launched, as the `app_control()` function is invoked in both cases of receiving notification and changing registration state.

```
#define EXTRA_DATA_FROM_REGISTRATION_CHANGE "registration_change"
#define EXTRA_DATA_FROM_NOTIFICATION "notification"

static void
app_control(app_control_h app_control, void *data)
{
    char *value = NULL;
    app_control_get_extra_data(app_control, APP_CONTROL_DATA_PUSH_LAUNCH_TYPE, &value);
    if (value) {
        if (!strcmp(value, EXTRA_DATA_FROM_NOTIFICATION))
            /* Add your code here when push messages arrive */
        else if (!strcmp(value, EXTRA_DATA_FROM_REGISTRATION_CHANGE))
            /* Add your code here when registration state is changed */
    }
}
```

- Store the notification at the push service database and request it later when the application is launched.

  You need to set the action to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

  You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched.

```
if (push_conn) {
    int ret = push_service_request_unread_notification(push_conn);
    if (ret != PUSH_SERVICE_ERROR_NONE)
        dlog_print(DLOG_ERROR, LOG_TAG, "ERROR: push_service_request_unread_notification()
failed.");
}
```

  The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification through the app control callback function.

- Discard the notification.

  You need to set the action to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service delivers the notification only when the application is up and running. Otherwise, the push service does not store the notification and discards it.

- Request unread notifications.

  If the user does not launch the application from the quick panel, the application requests the unread notifications after start-up using the asynchronous `push_service_request_unread_notification()` function.

  The following example shows a synchronous request using the `push_service_get_unread_notification()` function:

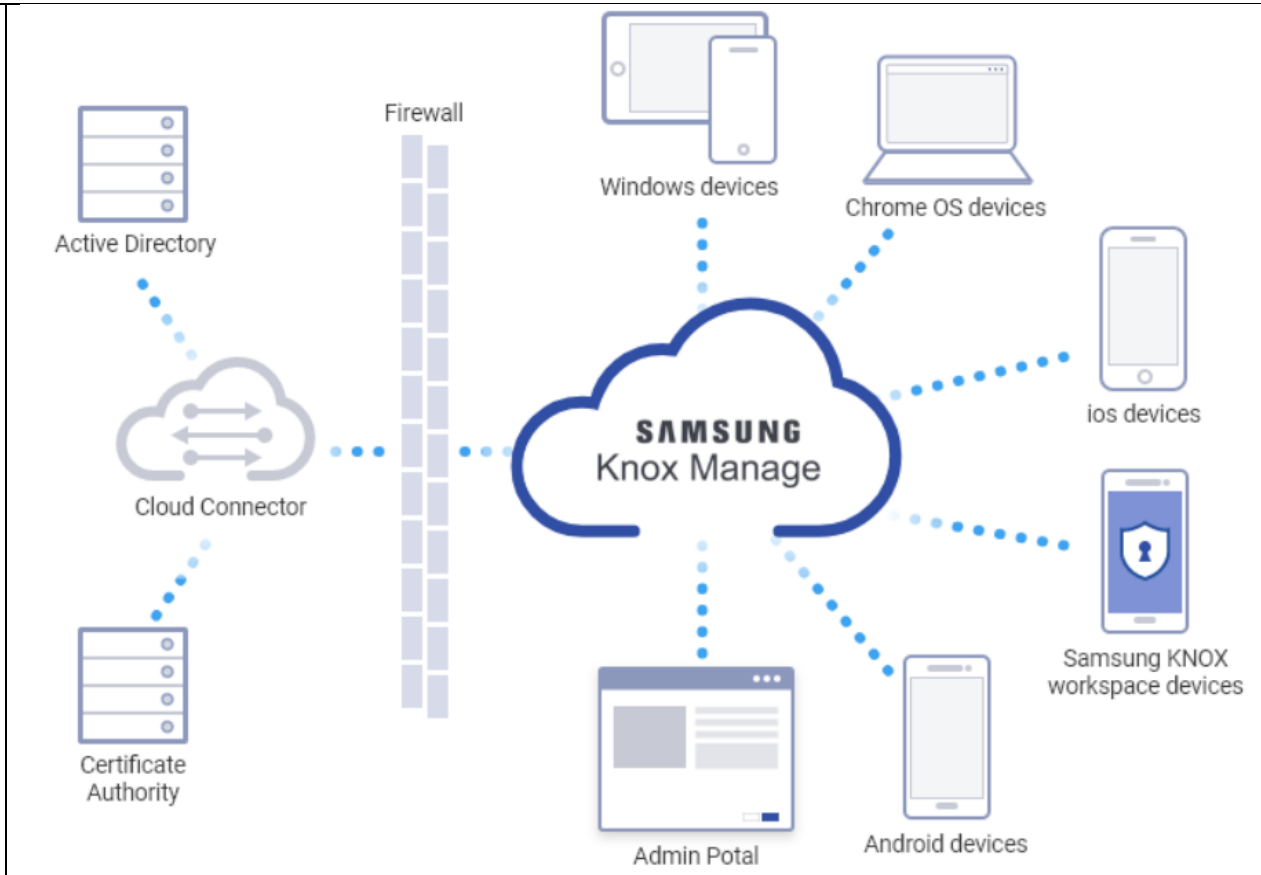
```
push_service_notification_h noti;
int ret;
do {
    ret = push_service_get_unread_notification(push_conn, &noti);

    /* Process the unread message noti */

    push_server_free_notification(&noti);
} while (1);
```

  Call this function repeatedly until no notification is returned. If there are multiple unread notifications, the notifications are retrieved in their arrival order.

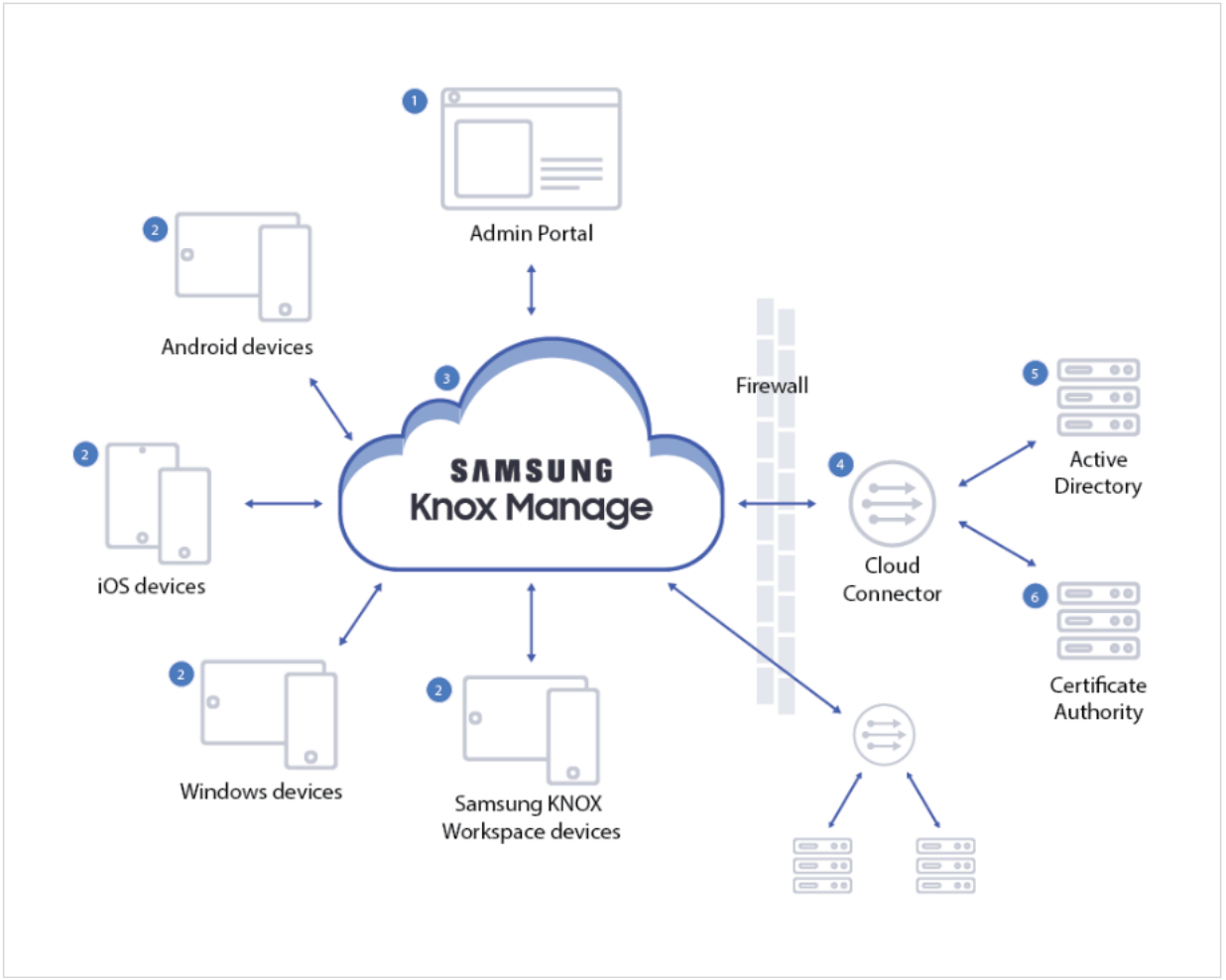
  The `push_service_get_unread_notification()` function blocks the code while it receives a notification from the service. Unless you need this kind of synchronous behavior, use the asynchronous function.

https://docs.tizen.org/application/native/guides/messaging/push/.

As another example, Samsung's Knox servers include memory to store content from received messages for delivery to a device.



https://docs.samsungknox.com/admin/knox-manage/welcome.htm



https://docs.samsungknox.com/admin/knox-manage/km-features.htm

Update an existing device profile

22.11 23.03 UAT

An admin can update the device's profile with a push update if a device is currently in a Configured state.

> **NOTE** — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile.

> **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not.

6. If displayed, select the **Push update without requesting end user consent** to push the updates directly to the configured state devices without consent from the device end user. Leave this option unselected affords the device user the ability to approve the profile update before its pushed to their device.
7. Select **YES** to proceed with the device push update and profile overwrite. Select **Back** to move back to the previous **Assign devices with profiles** screen.

https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm

	Components of Knox Manage 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	Set the profile update schedule You can set the schedule to update the latest policy applied to user devices on a regular basis. You can define multiple schedules. Click + to add a schedule. You can add or edit up to 20 schedules when you save the settings. To configure a policy update schedule: 1. Navigate to **Setting** > **Configuration** > **Profile Update Scheduler**. 2. Click next to Profile Update Schedule to enable the scheduler feature. 3. Select a target type from the following. • **Global Setting**—All profiles are updated according to the schedule. • **Set by Group / Organization**—You can configure multiple schedules and select groups or organizations for each schedule setting. 4. Select days and set the start time and time zone. • Select groups or organizations for each schedule setting if you selected the group/organization target type. • The policy update time may vary depending on the time and time zone of the user device. 5. Click **Save & Apply**. https://docs.samsungknox.com/admin/knox-manage/set-the-profile-update-schedule.htm.
[1e] the logic to determine when one of a plurality of message delivery triggers for the given one of the wireless end-user devices has occurred, wherein for at least some of the received network element messages, the receipt of such a message by the message buffer system is not a	The Accused Devices include "the logic to determine when one of a plurality of message delivery triggers for the given one of the wireless end-user devices has occurred, wherein for at least some of the received network element messages, the receipt of such a message by the message buffer system is not a message delivery trigger, and for at least one of the message delivery triggers, the trigger is an occurrence of an asynchronous event with time-critical messaging needs."

message delivery trigger, and for at least one of the message delivery triggers, the trigger is an occurrence of an asynchronous event with time-critical messaging needs, and	For example, Samsung's push messaging servers include logic (e.g., computer hardware, software, and instructions to determine when, how, and under what circumstances to store and/or deliver messages) to determine whether a delivery trigger that is not based on the receipt of the message (e.g., an event other than the arrival of the message) has occurred, such as when an event with time-critical messaging needs arises (e.g., when the application is running). *See e.g.*, When a notification arrives at the device, its delivery mechanism depends on whether the application is running: • When the application is running When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case. • When the application is not running If the notification arrives when the application is not running, there are 3 ways to handle the notification: ◦ Forcibly launch the application and deliver the notification to it. This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle. For more information, see the Retrieving Messages When Launched by the Push Service use case. ◦ Store the notification at the push service database and request it later when the application is launched. This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. For more information, see the Retrieving Missed Push Messages use case. The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification. ◦ Discard the notification. This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running. https://docs.tizen.org/application/web/guides/messaging/push/;

If the notification arrives when the application is not running, it can be handled in 3 ways:

- Forcibly launch the application and deliver the notification to it.

  You need to set the action to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

  When you create a project in the Tizen Studio, the `app_control()` function is created automatically. When the application is launched by another application or process (in this case, by the push service), all related information regarding this launch request is delivered through the `app_control` parameter. From this handle, retrieve the `op` operation using the `app_control_get_operation()` function. With `app_control` and `op`, retrieve the notification data using the `push_service_app_control_to_noti_data()` function.

  If the application is not launched by the push service, this function returns as `NULL`.

```
static void
app_control(app_control_h app_control, void *data)
{
    char *op = NULL;
    push_service_notification_h noti = NULL;
    int ret;

    if (app_control_get_operation(app_control, &op) < 0)
        return;

    /* Retrieve the noti from the bundle */
    ret = push_service_app_control_to_notification(app_control, op, &noti);

    if (noti) {
        /* Handle the noti */

        /* Free the noti */
        push_service_free_notification(noti);
    } else {
        /* Case when the application is not launched by the push service */
    }
    if (op)
        free(op);
}
```

Since Tizen 3.0, the push service provides launch types when the application is launched by the service. Use the following code to figure out why the application is launched, as the `app_control()` function is invoked in both cases of receiving notification and changing registration state.

```
#define EXTRA_DATA_FROM_REGISTRATION_CHANGE "registration_change"
#define EXTRA_DATA_FROM_NOTIFICATION "notification"

static void
app_control(app_control_h app_control, void *data)
{
    char *value = NULL;
    app_control_get_extra_data(app_control, APP_CONTROL_DATA_PUSH_LAUNCH_TYPE, &value);
    if (value) {
        if (!strcmp(value, EXTRA_DATA_FROM_NOTIFICATION))
            /* Add your code here when push messages arrive */
        else if (!strcmp(value, EXTRA_DATA_FROM_REGISTRATION_CHANGE))
            /* Add your code here when registration state is changed */
    }
}
```

- Store the notification at the push service database and request it later when the application is launched.

  You need to set the action to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

  You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched.

```
if (push_conn) {
    int ret = push_service_request_unread_notification(push_conn);
    if (ret != PUSH_SERVICE_ERROR_NONE)
        dlog_print(DLOG_ERROR, LOG_TAG, "ERROR: push_service_request_unread_notification()
failed.");
}
```

  The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification through the app control callback function.

- Discard the notification.

  You need to set the action to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service delivers the notification only when the application is up and running. Otherwise, the push service does not store the notification and discards it.

- Request unread notifications.

  If the user does not launch the application from the quick panel, the application requests the unread notifications after start-up using the asynchronous `push_service_request_unread_notification()` function.

  The following example shows a synchronous request using the `push_service_get_unread_notification()` function:

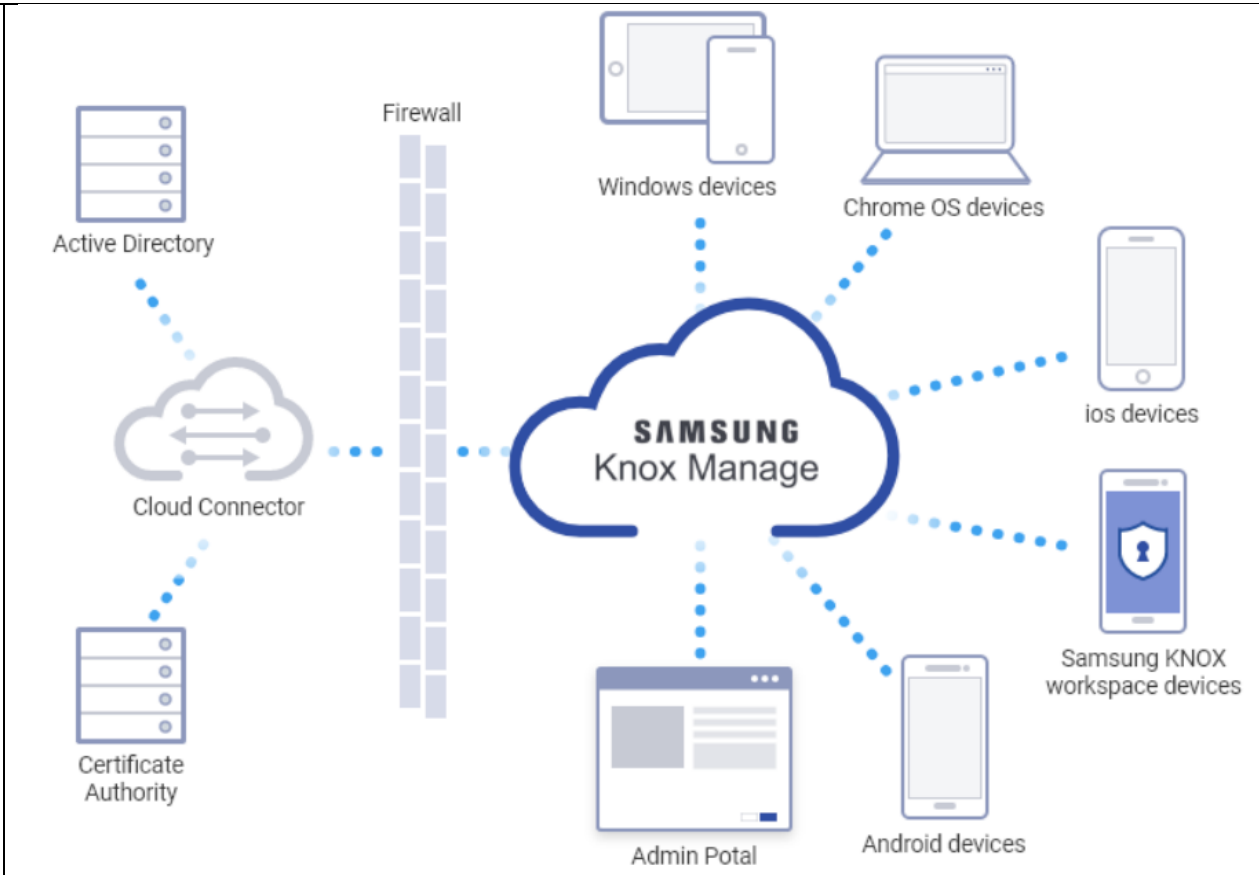
```
push_service_notification_h noti;
int ret;
do {
    ret = push_service_get_unread_notification(push_conn, &noti);

    /* Process the unread message noti */

    push_server_free_notification(&noti);
} while (1);
```

  Call this function repeatedly until no notification is returned. If there are multiple unread notifications, the notifications are retrieved in their arrival order.

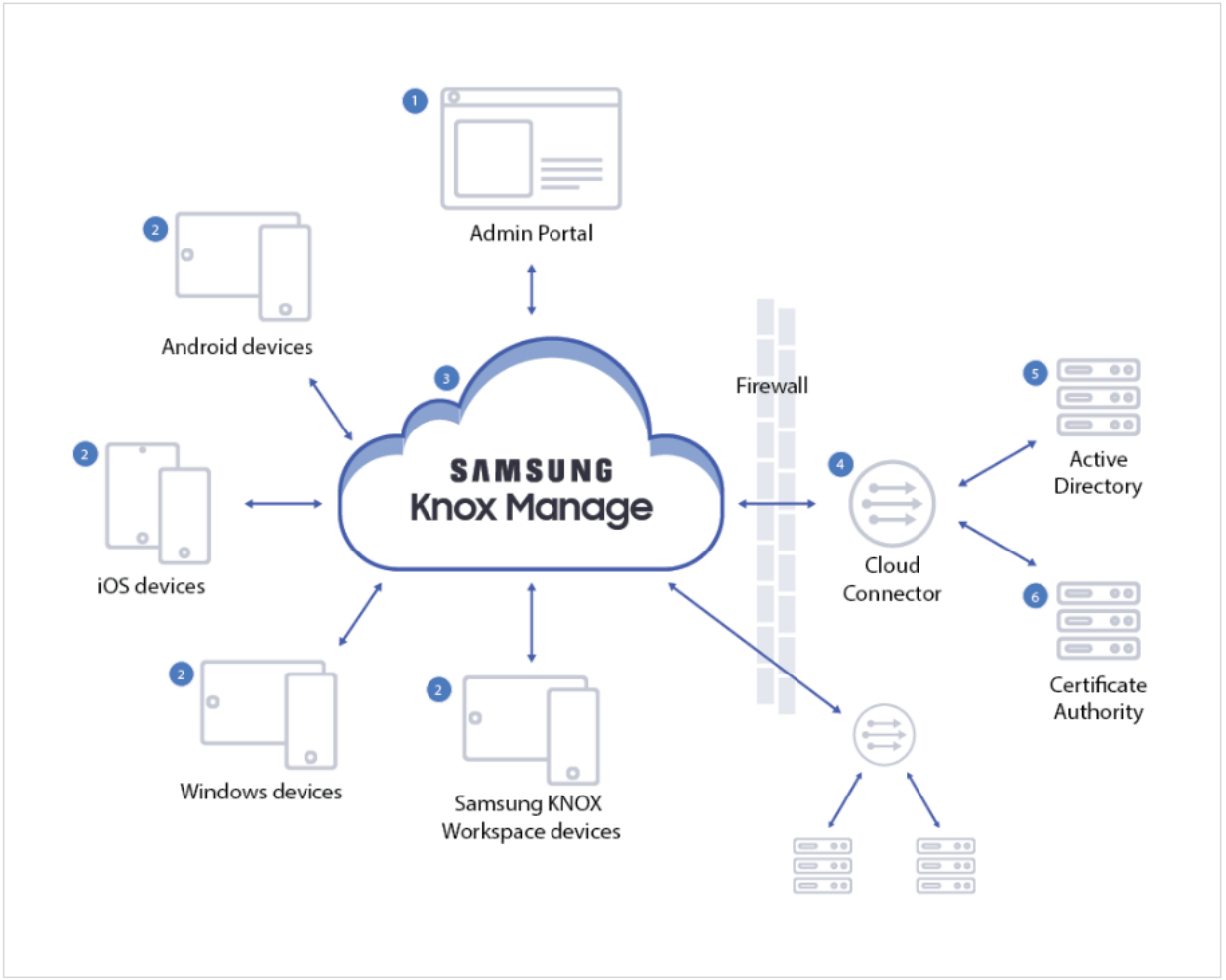
  The `push_service_get_unread_notification()` function blocks the code while it receives a notification from the service. Unless you need this kind of synchronous behavior, use the asynchronous function.

https://docs.tizen.org/application/native/guides/messaging/push/.

As another example, Samsung's Knox servers include logic (e.g., computer hardware, software, and instructions to determine when, how, and under what circumstances to store and/or deliver messages) to determine whether a delivery trigger that is not based on the receipt of the message (e.g., an event other than the arrival of the message) has occurred, such as when an event with time-critical messaging needs arises (e.g., when the application is running) .



https://docs.samsungknox.com/admin/knox-manage/welcome.htm



https://docs.samsungknox.com/admin/knox-manage/km-features.htm

Update an existing device profile

22.11 23.03 UAT

An admin can update the device's profile with a push update if a device is currently in a Configured state.

> **NOTE** — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile.

> **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not.

6. If displayed, select the **Push update without requesting end user consent** to push the updates directly to the configured state devices without consent from the device end user. Leave this option unselected affords the device user the ability to approve the profile update before its pushed to their device.
7. Select **YES** to proceed with the device push update and profile overwrite. Select **Back** to move back to the previous **Assign devices with profiles** screen.

https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm

	Components of Knox Manage 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm

Set the profile update schedule

You can set the schedule to update the latest policy applied to user devices on a regular basis.

You can define multiple schedules. Click + to add a schedule.

You can add or edit up to 20 schedules when you save the settings.

To configure a policy update schedule:

1. Navigate to **Setting** > **Configuration** > **Profile Update Scheduler**.
2. Click next to Profile Update Schedule to enable the scheduler feature.
3. Select a target type from the following.
   - **Global Setting**—All profiles are updated according to the schedule.
   - **Set by Group / Organization**—You can configure multiple schedules and select groups or organizations for each schedule setting.
4. Select days and set the start time and time zone.
   - Select groups or organizations for each schedule setting if you selected the group/organization target type.
   - The policy update time may vary depending on the time and time zone of the user device.
5. Click **Save & Apply**.

https://docs.samsungknox.com/admin/knox-manage/set-the-profile-update-schedule.htm.

As a further example, Samsung's Tizen servers and Knox servers are able to deliver commands and messages to devices (such as Tizen OS smartwatches and televisions), Knox-managed devices, and Knox administrator devices when an asynchronous event with time-critical messaging needs is detected within the network. Such events include, for example, the detection of specific events monitored by software at a device (e.g., https://docs.samsungknox.com/admin/knox-manage/configure-alerts.htm, and purchase or billing transactions initiated by one device). As another example, an asynchronous event with time-critical messaging needs may be the detection of a device as having been connected to the

	Internet, Knox servers, or Tizen servers, in which case messages intended for that device are delivered to the device at that point.
[1f] upon determining that one of the message delivery triggers has occurred, the logic further to supply one or more messages comprising the buffered content to the transport services stack for delivery on the secure message link maintained between the transport services stack and a device link agent on the given one of the wireless end-user devices.	The Accused Devices include "upon determining that one of the message delivery triggers has occurred, the logic further to supply one or more messages comprising the buffered content to the transport services stack for delivery on the secure message link maintained between the transport services stack and a device link agent on the given one of the wireless end-user devices." For example, Samsung's push messaging servers comprise logic (e.g., computer hardware, software, and instructions to determine when, how, and under what circumstances to store and/or deliver messages) which supplies buffered content for delivery via the transport services stack (e.g., hardware and software responsible for communicating data) on a secure message link (e.g., a secure network connection) between the transport services stack and a device link agent (operating system software or applications on the device) on the device. *See e.g.*,

When a notification arrives at the device, its delivery mechanism depends on whether the application is running:

- When the application is running

  When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case.

- When the application is not running

  If the notification arrives when the application is not running, there are 3 ways to handle the notification:

  - Forcibly launch the application and deliver the notification to it.

    This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

    For more information, see the Retrieving Messages When Launched by the Push Service use case.

  - Store the notification at the push service database and request it later when the application is launched.

    This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

    For more information, see the Retrieving Missed Push Messages use case.

    The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

  - Discard the notification.

    This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

https://docs.tizen.org/application/web/guides/messaging/push/;

If the notification arrives when the application is not running, it can be handled in 3 ways:

- Forcibly launch the application and deliver the notification to it.

  You need to set the action to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

  When you create a project in the Tizen Studio, the `app_control()` function is created automatically. When the application is launched by another application or process (in this case, by the push service), all related information regarding this launch request is delivered through the `app_control` parameter. From this handle, retrieve the `op` operation using the `app_control_get_operation()` function. With `app_control` and `op`, retrieve the notification data using the `push_service_app_control_to_noti_data()` function.

  If the application is not launched by the push service, this function returns as `NULL`.

```
static void
app_control(app_control_h app_control, void *data)
{
    char *op = NULL;
    push_service_notification_h noti = NULL;
    int ret;

    if (app_control_get_operation(app_control, &op) < 0)
        return;

    /* Retrieve the noti from the bundle */
    ret = push_service_app_control_to_notification(app_control, op, &noti);

    if (noti) {
        /* Handle the noti */

        /* Free the noti */
        push_service_free_notification(noti);
    } else {
        /* Case when the application is not launched by the push service */
    }
    if (op)
        free(op);
}
```

Since Tizen 3.0, the push service provides launch types when the application is launched by the service. Use the following code to figure out why the application is launched, as the `app_control()` function is invoked in both cases of receiving notification and changing registration state.

```
#define EXTRA_DATA_FROM_REGISTRATION_CHANGE "registration_change"
#define EXTRA_DATA_FROM_NOTIFICATION "notification"

static void
app_control(app_control_h app_control, void *data)
{
    char *value = NULL;
    app_control_get_extra_data(app_control, APP_CONTROL_DATA_PUSH_LAUNCH_TYPE, &value);
    if (value) {
        if (!strcmp(value, EXTRA_DATA_FROM_NOTIFICATION))
            /* Add your code here when push messages arrive */
        else if (!strcmp(value, EXTRA_DATA_FROM_REGISTRATION_CHANGE))
            /* Add your code here when registration state is changed */
    }
}
```

- Store the notification at the push service database and request it later when the application is launched.

  You need to set the action to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

  You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched.

```
if (push_conn) {
    int ret = push_service_request_unread_notification(push_conn);
    if (ret != PUSH_SERVICE_ERROR_NONE)
        dlog_print(DLOG_ERROR, LOG_TAG, "ERROR: push_service_request_unread_notification()
failed.");
}
```

  The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification through the app control callback function.

- Discard the notification.

  You need to set the action to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service delivers the notification only when the application is up and running. Otherwise, the push service does not store the notification and discards it.

- Request unread notifications.

  If the user does not launch the application from the quick panel, the application requests the unread notifications after start-up using the asynchronous `push_service_request_unread_notification()` function.

  The following example shows a synchronous request using the `push_service_get_unread_notification()` function:

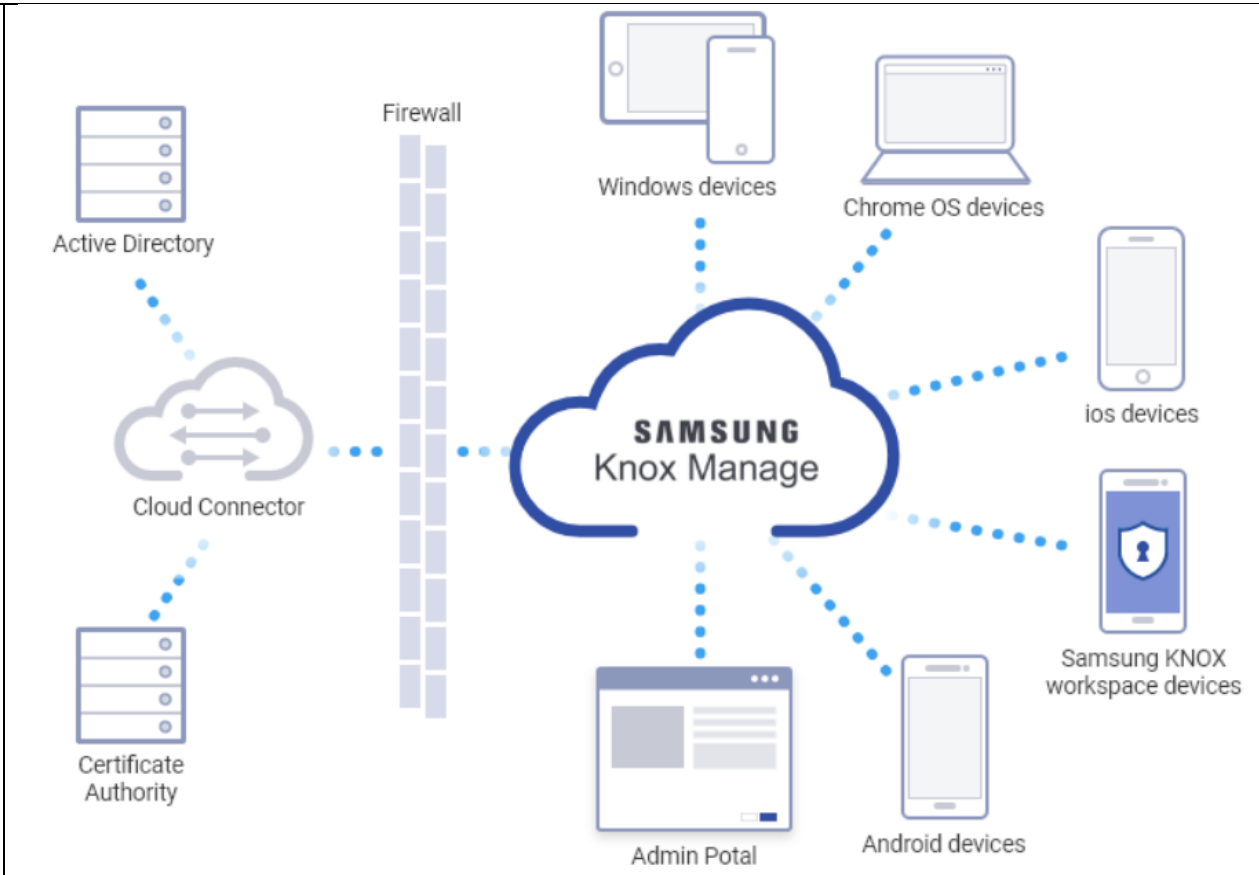
```
push_service_notification_h noti;
int ret;
do {
    ret = push_service_get_unread_notification(push_conn, &noti);

    /* Process the unread message noti */

    push_server_free_notification(&noti);
} while (1);
```

  Call this function repeatedly until no notification is returned. If there are multiple unread notifications, the notifications are retrieved in their arrival order.

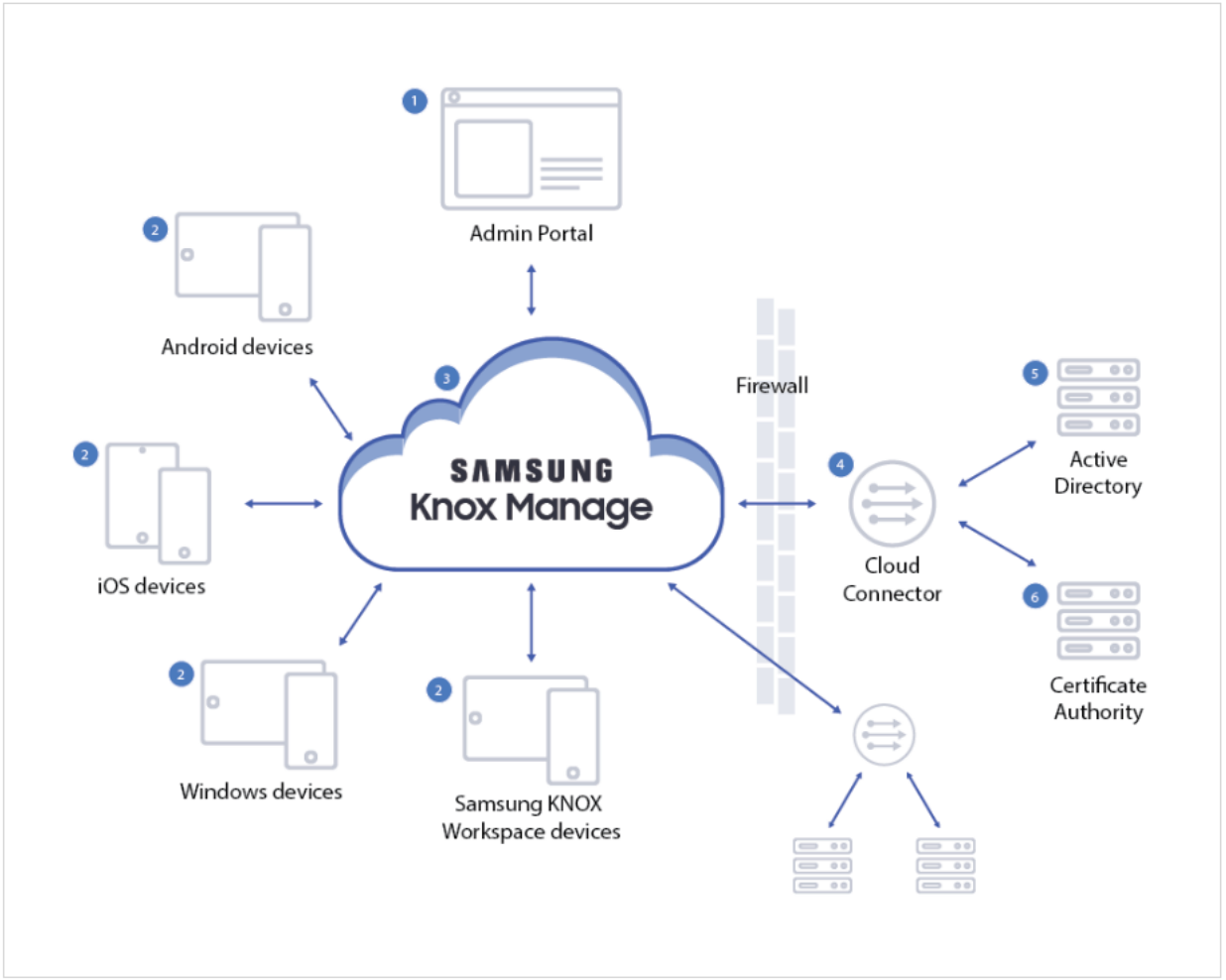
  The `push_service_get_unread_notification()` function blocks the code while it receives a notification from the service. Unless you need this kind of synchronous behavior, use the asynchronous function.

https://docs.tizen.org/application/native/guides/messaging/push/.

As another example, Samsung's Knox servers comprise logic (e.g., computer hardware, software, and instructions to determine when, how, and under what circumstances to store and/or deliver messages) which supplies buffered content for delivery via the transport services stack (e.g., hardware and software responsible for communicating data) on a secure message link (e.g., a secure network connection) between the transport services stack and a device link agent (operating system software or applications on the device) on the device.



https://docs.samsungknox.com/admin/knox-manage/welcome.htm



https://docs.samsungknox.com/admin/knox-manage/km-features.htm

Update an existing device profile

22.11 23.03 UAT

An admin can update the device's profile with a push update if a device is currently in a Configured state.

> **NOTE** — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile.

> **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not.

6. If displayed, select the **Push update without requesting end user consent** to push the updates directly to the configured state devices without consent from the device end user. Leave this option unselected affords the device user the ability to approve the profile update before its pushed to their device.
7. Select **YES** to proceed with the device push update and profile overwrite. Select **Back** to move back to the previous **Assign devices with profiles** screen.

https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm

	Components of Knox Manage 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm

Set the profile update schedule

You can set the schedule to update the latest policy applied to user devices on a regular basis.

You can define multiple schedules. Click + to add a schedule.

You can add or edit up to 20 schedules when you save the settings.

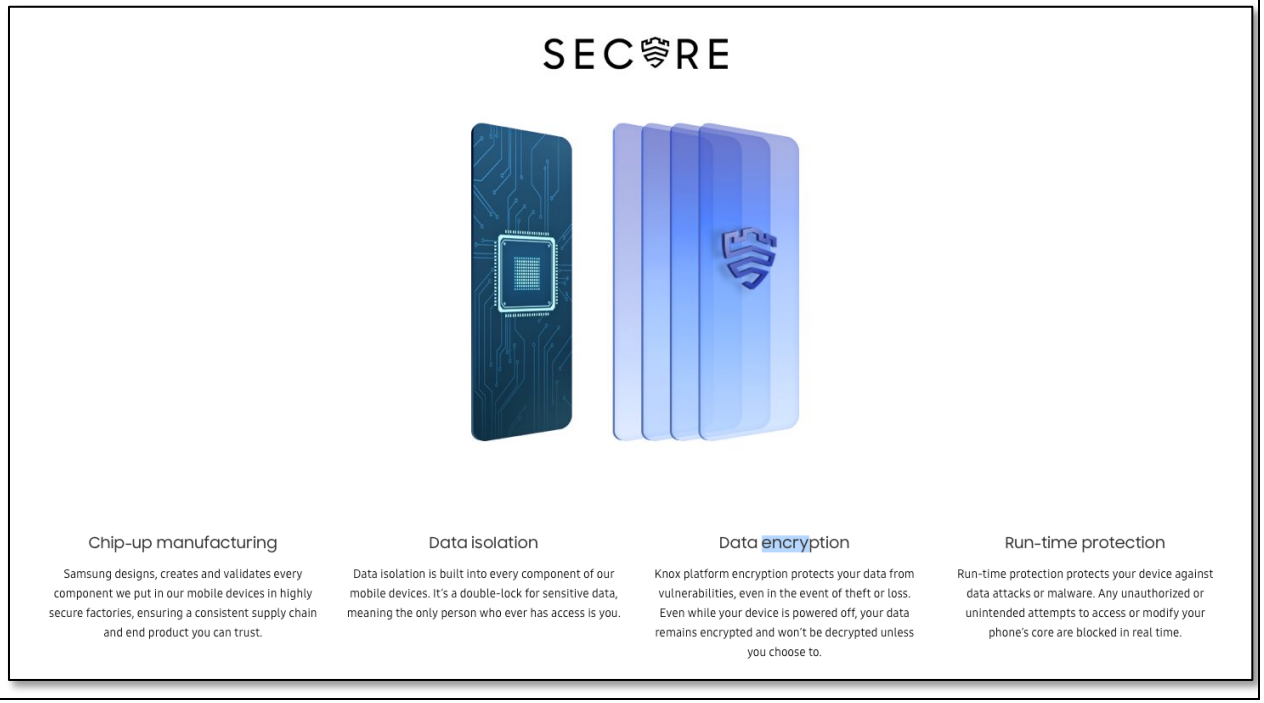
To configure a policy update schedule:

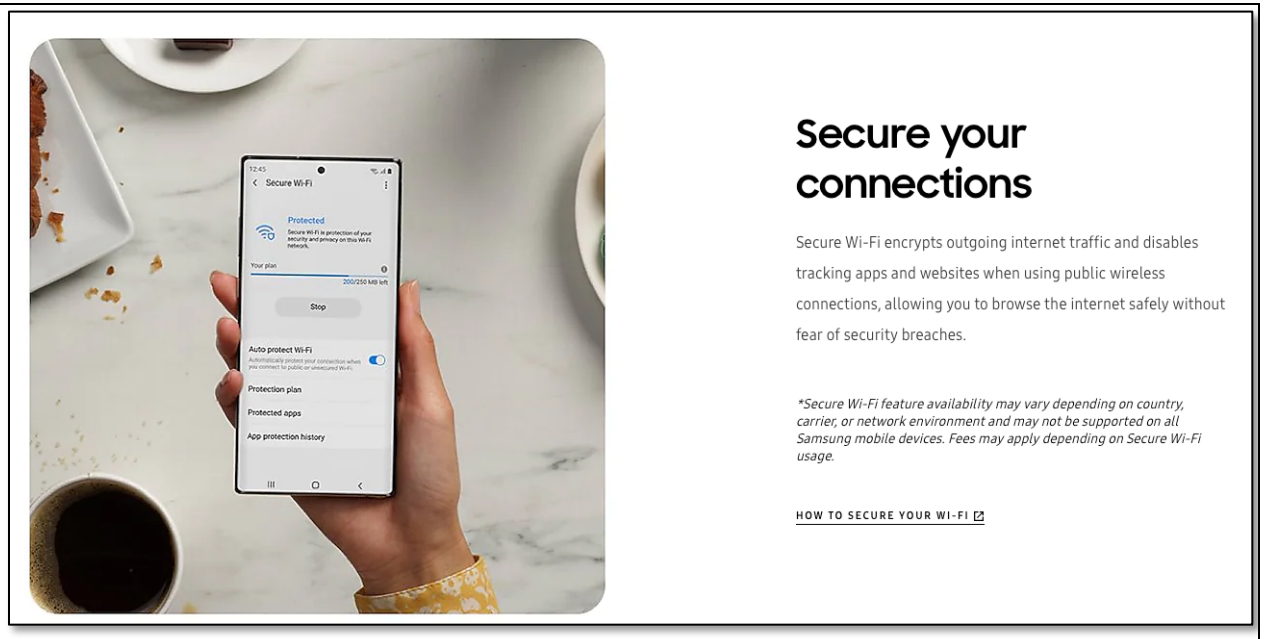
1. Navigate to **Setting** > **Configuration** > **Profile Update Scheduler**.
2. Click next to Profile Update Schedule to enable the scheduler feature.
3. Select a target type from the following.
   - **Global Setting**—All profiles are updated according to the schedule.
   - **Set by Group / Organization**—You can configure multiple schedules and select groups or organizations for each schedule setting.
4. Select days and set the start time and time zone.
   - Select groups or organizations for each schedule setting if you selected the group/organization target type.
   - The policy update time may vary depending on the time and time zone of the user device.
5. Click **Save & Apply**.

https://docs.samsungknox.com/admin/knox-manage/set-the-profile-update-schedule.htm.

As a further example, Samsung's Tizen and Knox products utilize communication layers comprising a transport services stack at the servers and devices to which the servers are connected. As one example, the secure control link between Knox servers and Knox-managed devices communicate through specific software residing on both Knox servers (Knox Manage software) and Knox-managed devices (e.g., Knox agent at the devices) which receives, decrypts and decodes, interprets, and executes commands or messages from other elements in the system. Likewise, with Samsung's Tizen devices, specific software within both Tizen OS and Tizen's push messaging servers operate in a similar fashion.

Claim 2

2. The message link server of claim 1, further comprising an encrypt function to encrypt the one or more messages supplied to the transport services stack for delivery on the secure message link maintained between the message link server and the device link agent on the given one of the wireless end-user devices.	The Accused Devices comprise the "message link server of claim 1, further comprising an encrypt function to encrypt the one or more messages supplied to the transport services stack for delivery on the secure message link maintained between the message link server and the device link agent on the given one of the wireless end-user devices." For example, the message link servers of the accused instrumentalities use encryption protocols to secure communications between the device and the network system. *See e.g.,* 



https://www.samsung.com/us/security/;

Firebase > Documentation > FCM > Engage

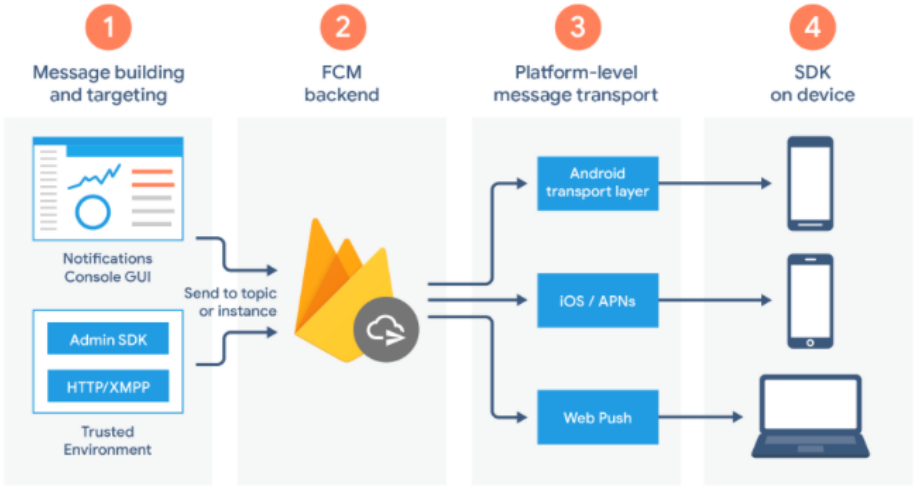
Was this helpful?

FCM Architectural Overview

Send feedback

FCM relies on the following set of components that build, transport, and receive messages:

1. Tooling to compose or build message requests. The Notifications composer provides a GUI-based option for creating notification requests. For full automation and support for all message types, you must build message requests in a trusted server environment that supports the Firebase Admin SDK or the FCM server protocols. This environment could be Cloud Functions for Firebase, App Engine, or your own app server.



2. The FCM backend, which (among other functions) accepts message requests, performs fanout of messages via topics, and generates message metadata such as the message ID.
3. A platform-level transport layer, which routes the message to the targeted device, handles message delivery, and applies platform-specific configuration where appropriate. This transport layer includes:
   - Android transport layer (ATL) for Android devices with Google Play services
   - Apple Push Notification service (APNs) for Apple devices
   - Web push protocol for web apps

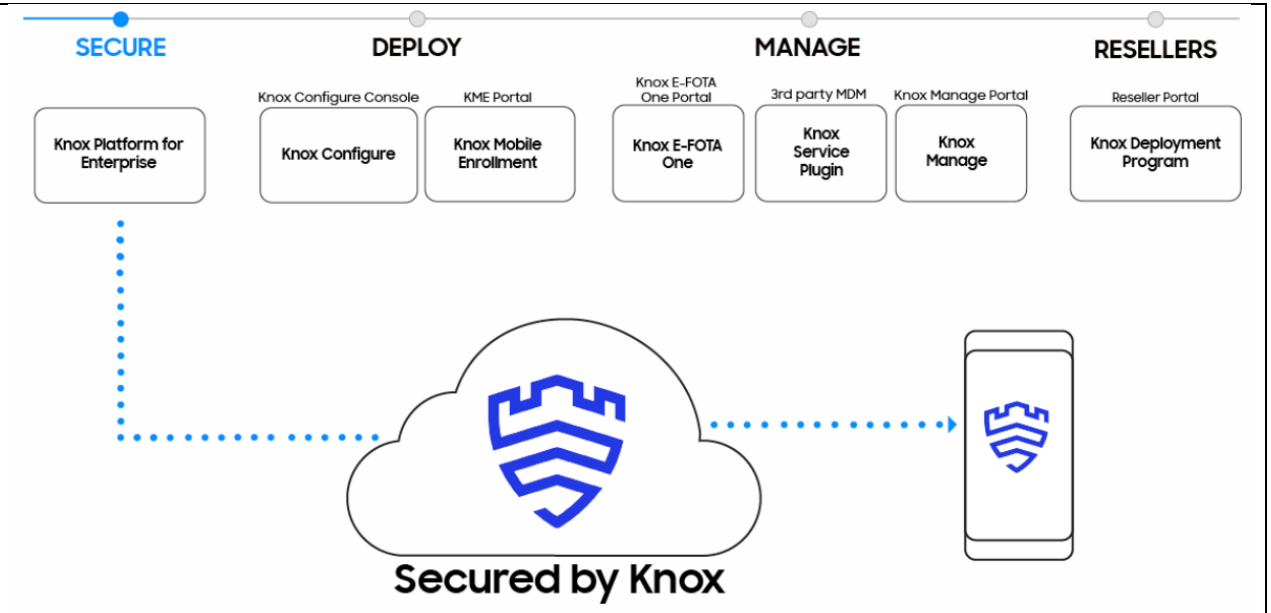
   ★ **Note:** Platform-level transport layers are outside the core FCM product. FCM messages routed to a platform-level transport layer may be subject to terms specific to that platform rather than FCM's terms of service. Android message routing via ATL falls under the Google APIs terms of service.
4. The FCM SDK on the user's device, where the notification is displayed or the message is handled according to the app's foreground/background state and any relevant application logic.

https://firebase.google.com/docs/cloud-messaging/fcm-architecture;

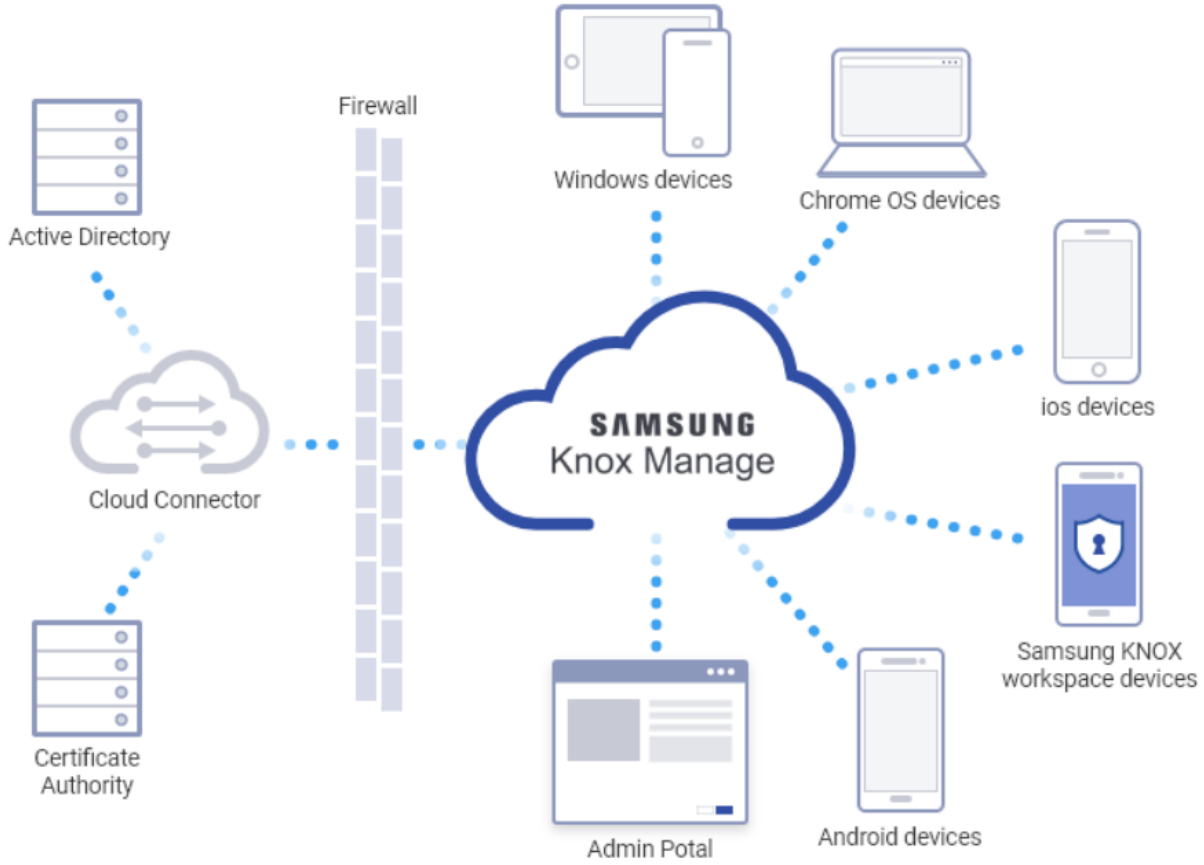
Encryption for data messages

The Android Transport Layer (see FCM architecture) uses point-to-point encryption. Depending on your needs, you may decide to add end-to-end encryption to data messages. FCM does not provide an end-to-end solution. However, there are external solutions available such as Capillary or DTLS.

https://firebase.google.com/docs/cloud-messaging/concept-options#encryption_for_data_messages.



https://docs.samsungknox.com/admin/fundamentals/welcome.htm



https://docs.samsungknox.com/admin/knox-manage/welcome.htm

The messages passed between servers of the accused instrumentalities and end user devices are also secured. As another example, the Tizen Push Server receives encrypted information from app servers and transmits such encrypted information on to device messaging agents residing on user devices, which in turn decrypt received messages.

Managing Security

When you send a notification with sensitive information, be aware of the chance that the notification gets hijacked by someone else. It is your responsibility to keep such sensitive information safe from malicious access. The following rules are strongly recommended:

- Keep the push application ID confidential.

  If the application ID is exposed, hackers can try to hijack notifications using a fake application with the exposed ID.

- Do not store the registration ID on the device.

  The registration ID can be considered as the destination address for notifications. Without the ID, hackers cannot send fake notifications to your application.

- Encrypt sensitive information.

  When you send sensitive information, such as personal information and financial transactions, encrypt it and load it to the notification as a payload instead of the message field. When the notification arrives at the device, the application decrypts the payload and retrieves the sensitive information.

- Do not hardcode the AppSecret in the source code.

  The AppSecret is a key to accessing the push server for sending notifications. If notifications are sent from your application server, the application does not need to know the AppSecret at all. Keep the AppSecret in the server and do not load any related information in the application. If you want device-to-device notification delivery without your application server, the application needs the AppSecret to send a notification from a device. In this case, it is your responsibility to keep the AppSecret safe.

Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the Push Server guide for server developers.

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

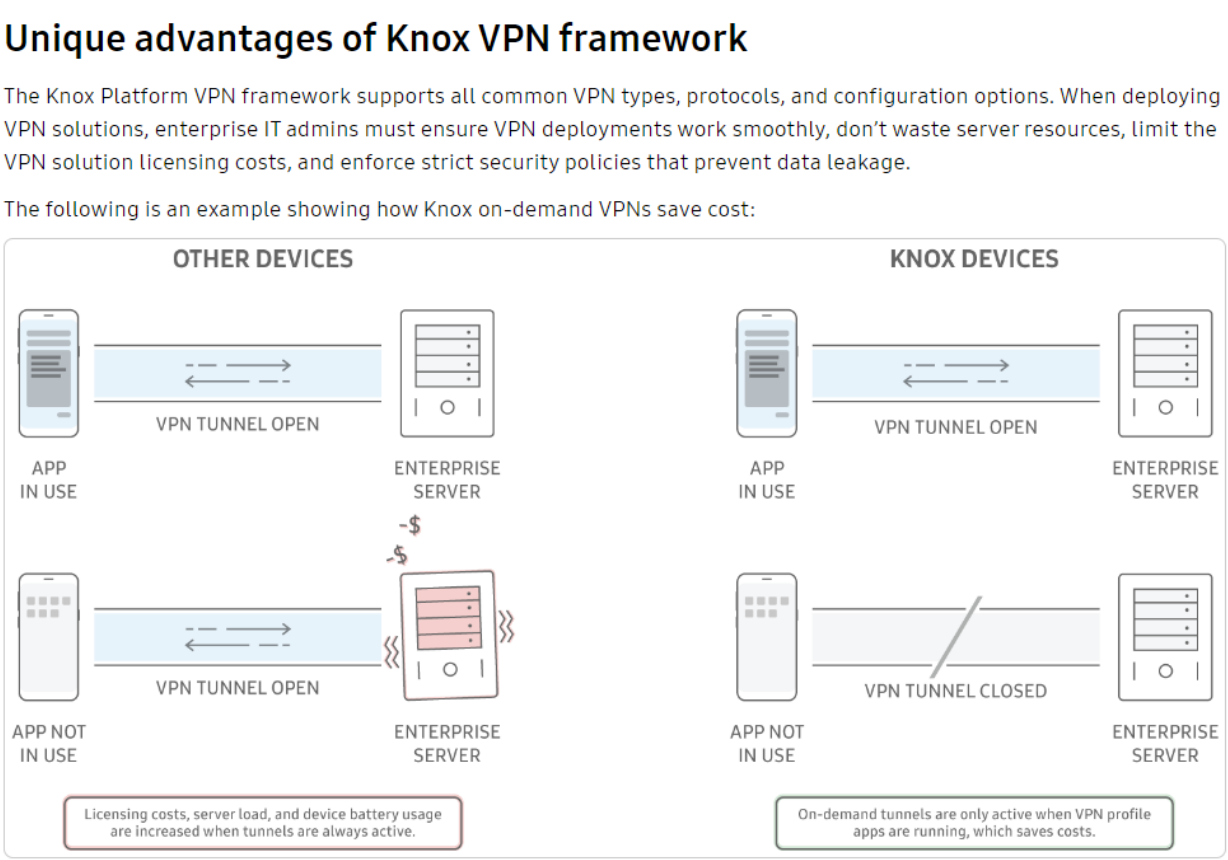
- Body:

```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

https://docs.tizen.org/application/native/guides/messaging/push/#managing-security

As another example, the Firebase framework encrypts messages in transit. See: https://firebase.google.com/docs/cloud-messaging/concept-options ("Encryption for data messages The Android Transport Layer (see FCM architecture) uses point-to-point encryption.").

Likewise, messages sent with Knox Manage instrumentalities are also encrypted. See https://docs.samsungknox.com/admin/fundamentals/whitepaper/core-platform-security/knox-vault/; https://docs.samsungknox.com/admin/fundamentals/whitepaper/network-security/virtual-private-networks/.



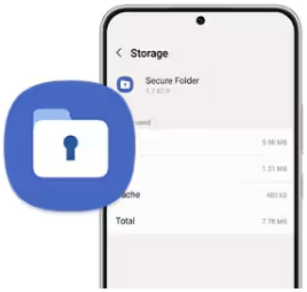
Claim 3

3. The message link server of claim 2, wherein the encrypted one or more messages are transported to the device link agent on the given one of the wireless end-user devices using one or more of encryption on the transport services stack, IP (Internet Protocol) layer encryption, and tunneling.	The accused instrumentalities comprise the "message link server of claim 2, wherein the encrypted one or more messages are transported to the device link agent on the given one of the wireless end-user devices using one or more of encryption on the transport services stack, IP (Internet Protocol) layer encryption, and tunneling." *See supra* claims 1 and 2.

Claim 4

4. The message link server of claim 1, wherein the device link agent executes in a secure execution environment on at least one of the devices, and at least one of the software components executes outside of the secure execution environment on that device.	The accused instrumentalities comprise the "message link server of claim 1, wherein the device link agent executes in a secure execution environment on at least one of the devices, and at least one of the software components executes outside of the secure execution environment on that device." *See supra* claim 1. As just one example, with Knox-managed devices, the Knox agent (e.g., a "device link agent") software residing on the user device executes from a specially protected partition of local memory. This specially protected partition is sometimes referred to as the Knox Secure Folder. Applications stored in the Secure Folder partition are managed separately with respect to push notifications. Other applications on the user device may be run from outside of the Knox Secure Folder partition.

What is the Secure Folder and how do I use it?



The Secure Folder is the perfect place to store all the photos, videos, files, apps and data that you want to keep private. The folder is protected by the defence-grade Samsung Knox ↗ security platform, which encrypts all data stored there, making sure that your information is kept safe from any malicious attacks. You can also add a passcode or biometric lock to keep your data safe from any prying eyes.

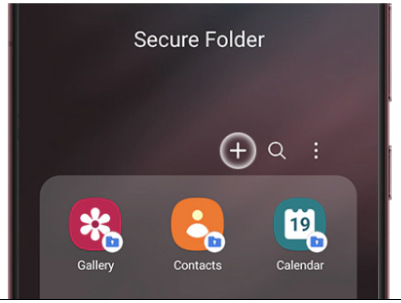
https://www.samsung.com/uk/support/mobile-devices/what-is-the-secure-folder-and-how-do-i-use-it/

Add apps or files in Secure Folder

You want to download your bank's app, but you're nervous someone will steal your information. If you add the app to Secure Folder, no one will even know it is on your phone except you.

After Secure Folder is set up, tap **Add apps** (the plus sign). Tap a desired app to put in the folder, and then tap **Add**. You'll see a copy of the app in your Secure Folder.

To add files, tap **More options** (the three vertical dots) and then tap **Add files**. You can now choose the type of file, or use My Files to find any file on your device.



	https://www.samsung.com/us/support/answer/ANS00081123/

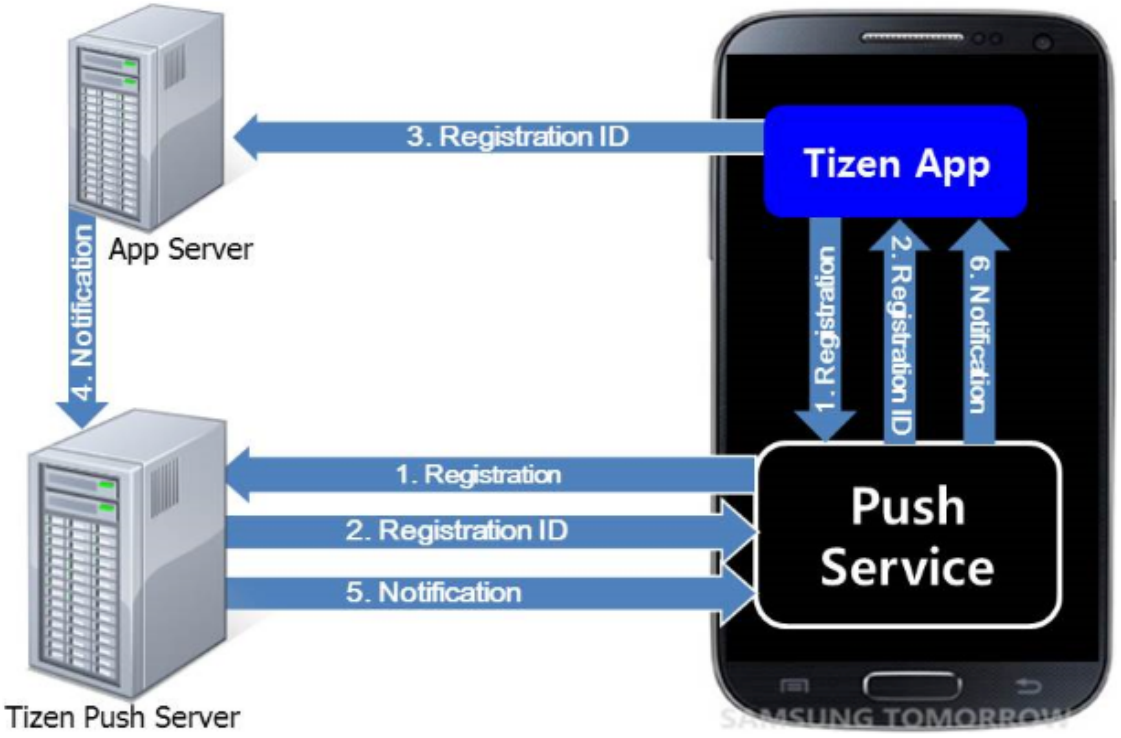
Claim 5

5. The message link server of claim 1, wherein the transport services stack is further to receive, over each of the respective secure message links, upload messages forwarded by the respective device link agents from at least a subset of the device software components, each of the upload messages identifying a corresponding one of the network elements to which the device respective software component has requested delivery, the network server system using the interface to a network to deliver content from the upload messages to the respective identified network elements.	The accused instrumentalities comprise the "message link server of claim 1, wherein the transport services stack is further to receive, over each of the respective secure message links, upload messages forwarded by the respective device link agents from at least a subset of the device software components, each of the upload messages identifying a corresponding one of the network elements to which the device respective software component has requested delivery, the network server system using the interface to a network to deliver content from the upload messages to the respective identified network elements." *See supra* claim 1. For example, the end user devices of the accused instrumentalities cause their respective device link agents of a subset of the software components on the device to send messages to servers such as the App Servers (e.g., from Tizen devices), Knox Manage servers (from Knox managed devices), and push messaging servers (e.g., from Samsung's Android devices). These messages include those for sending authentication and verification information to servers.

Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

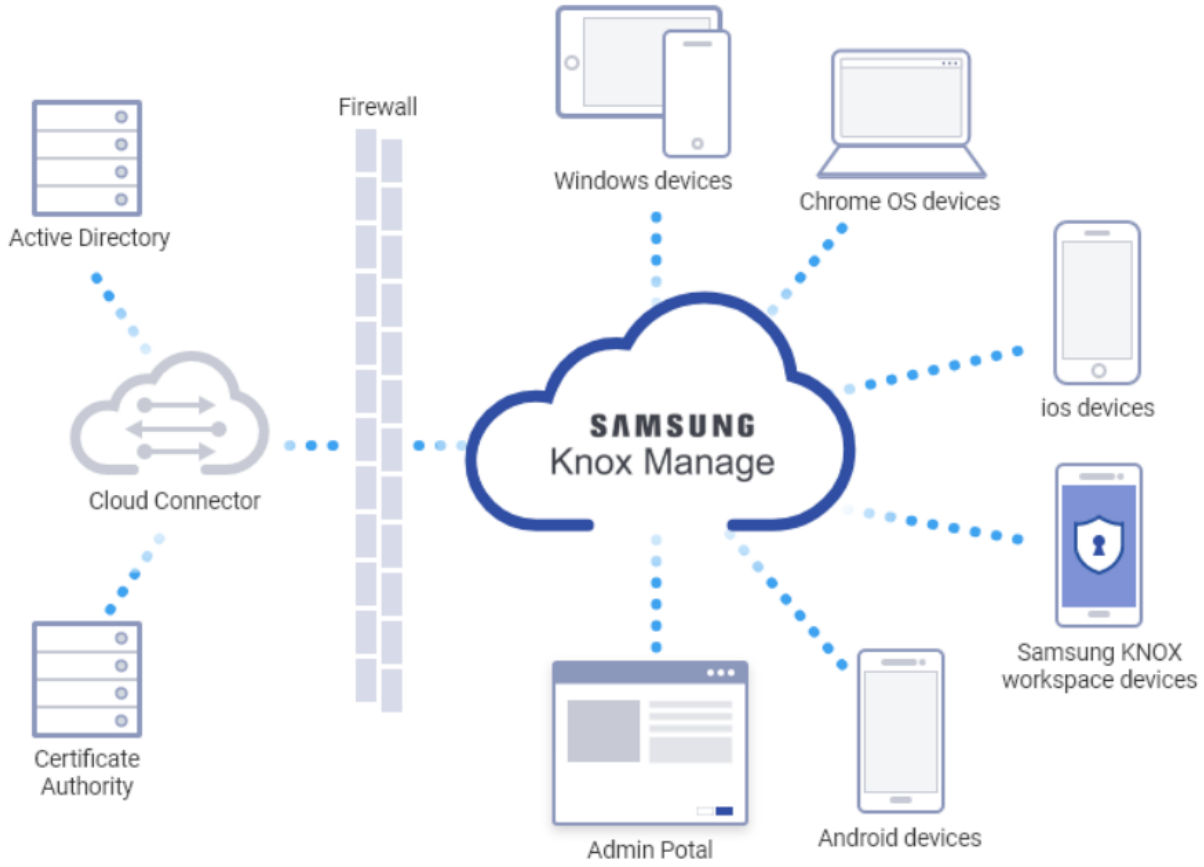
Figure: Service architecture



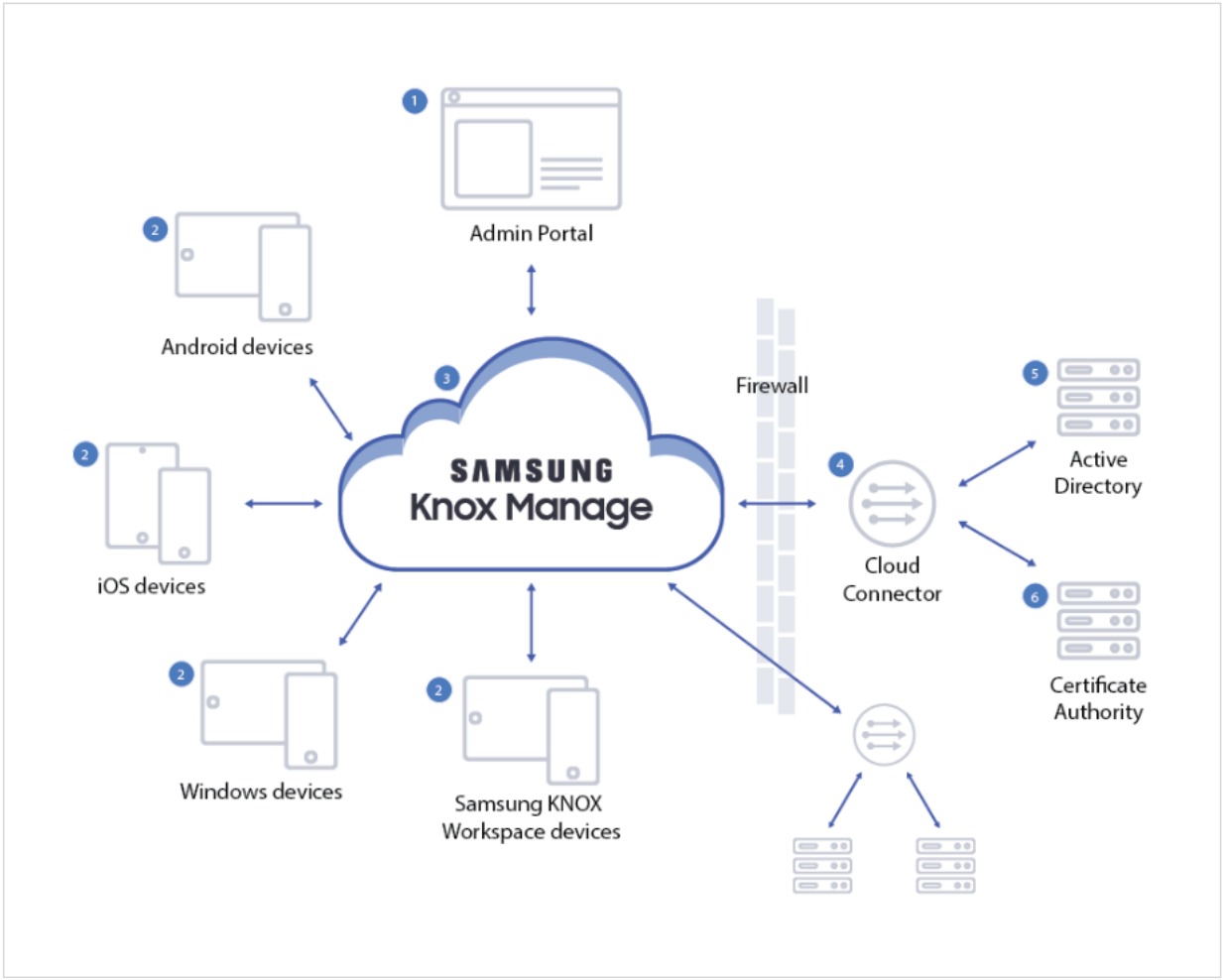
To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

https://docs.tizen.org/application/web/guides/messaging/push/;



https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	 https://docs.samsungknox.com/admin/knox-manage/km-features.htm

Claim 6

6. The message link server of claim 1, wherein at least one of the one or more messages for delivery by the transport	The accused instrumentalities comprise a "message link server of claim 1, wherein at least one of the one or more messages for delivery by the transport services stack comprises multiple identifier/data pairs."

services stack comprises multiple identifier/data pairs.	*See supra* claim 1. See, for example, https://firebase.google.com/docs/cloud-messaging/concept-options ("Data messages, by contrast, contain only your user-defined custom key-value pairs. Notification messages can contain an optional data payload. Maximum payload for both message types is 4000 bytes, except when sending messages from the Firebase console, which enforces a 1000 character limit."); https://docs.tizen.org/application/native/guides/messaging/push/ ("The message field contains not only the message to show in the quick panel on the device, but also the behaviors that the device must take when receiving the notification. The message field is a string that consists of key-value pairs. The available pair options are given in the following table."). On information and belief, messages transmitted to and from Knox servers also comprise multiple identifier/data pairs.

Claim 7

7. The message link server of claim 1, the device messaging agent on at least one of the wireless end-user devices further to initiate the respective secure Internet data message link to the transport services stack.	The accused instrumentalities comprise a "message link server of claim 1, the device messaging agent on at least one of the wireless end-user devices further to initiate the respective secure Internet data message link to the transport services stack." *See supra* claim 1, 5. As explained above, the device messaging agents on the devices may initiate secure connections to both app servers and the push messaging servers and Knox Manage servers to, inter alia, request push messages and other information and initiate authentication procedures.

Claim 8

8. The message link server of claim 1, further comprising a secure server to provide secure authorization signatures to the given one of the wireless end-user devices, the secure authorization signatures indicating the authorized software components that are allowed to	The accused instrumentalities comprise the "message link server of claim 1, further comprising a secure server to provide secure authorization signatures to the given one of the wireless end-user devices, the secure authorization signatures indicating the authorized software components that are allowed to receive data from secure message link messages via the message link server." *See supra* claim 1, 5. As explained above, the device messaging agents on the devices may initiate secure connections to both app servers and the push messaging servers and Knox Manage servers to, inter alia, request push messages and other information and initiate

receive data from secure message link messages via the message link server.	authentication procedures. Among these communications are transmissions from apps in the user devices to push messaging servers to register said applications as being permitted to receive messages and data from the servers. Likewise, Knox Manage software components on the user device communicate with Knox servers to authenticate the software on the user device to, among other things, allow the device to access Knox-related functionality over the network.

Claim 9

9. The message link server of claim 1, wherein one of the message delivery triggers is the expiration of a periodic timer.	The accused instrumentalities comprise the "message link server of claim 1, wherein one of the message delivery triggers is the expiration of a periodic timer." As explained in claim 1 *supra*, various triggers cause transmissions to occur between devices and push messaging servers and Knox servers. The transmission trigger may be a time period of inactivity or the expiration of a periodic timer. *See, e.g.*, https://docs.tizen.org/application/web/guides/messaging/push/: ▯ "Store the notification at the push service database and request it later when the application is launched. This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. For more information, see the Retrieving Missed Push Messages use case. The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

	Store the notification at the push service database and request it later when the application is launched. You need to set the action to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched. ▯ Request the unread notifications asynchronously. If there is such a notification, it can be received through the `_noti_cb()` function after the `_on_state_registered()` function returns. Once the request for unread notifications is successfully delivered, `PUSH_SERVICE_ERROR_NONE` is returned."

Claim 11

11. The message link server of claim 1, wherein one of the message delivery triggers is the receipt of a transmission on the respective secure message link from the device link agent of the given one of the wireless end-user devices, or a response generated to a transmission received from that device link agent.	The accused instrumentalities comprise the "message link server of claim 1, wherein one of the message delivery triggers is the receipt of a transmission on the respective secure message link from the device link agent of the given one of the wireless end-user devices, or a response generated to a transmission received from that device link agent." As explained in claim 1 *supra*, various triggers cause transmissions to occur between devices and push messaging servers and Knox servers. The transmission trigger may be the receipt by the messaging server of a notification from a user device that is has started up and/or is requesting push messages. *See, e.g.*, https://docs.tizen.org/application/web/guides/messaging/push/:

	▯ “Store the notification at the push service database and request it later when the application is launched. This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. For more information, see the Retrieving Missed Push Messages use case. The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification. Store the notification at the push service database and request it later when the application is launched. You need to set the action to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched. ▯ Request the unread notifications asynchronously. If there is such a notification, it can be received through the `_noti_cb()` function after the `_on_state_registered()` function returns. Once the request for unread notifications is successfully delivered, `PUSH_SERVICE_ERROR_NONE` is returned.”

Claim 12

12. The message link server of claim 11, wherein the transmission is a heartbeat message generated by the given device link agent, or a request received from the given device link agent.	The accused instrumentalities comprise the "message link server of claim 11, wherein the transmission is a heartbeat message generated by the given device link agent, or a request received from the given device link agent." As explained in claim 1 *supra*, various triggers cause transmissions to occur between devices and push messaging servers and Knox servers. The transmission trigger may be the receipt by the messaging server of a notification from a user device that is has started up and/or is requesting push messages. *See, e.g.*, https://docs.tizen.org/application/web/guides/messaging/push/: ▯ "Store the notification at the push service database and request it later when the application is launched. This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. For more information, see the Retrieving Missed Push Messages use case. The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification. Store the notification at the push service database and request it later when the application is launched.	

	You need to set the action to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched. ▯ Request the unread notifications asynchronously. If there is such a notification, it can be received through the `_noti_cb()` function after the `_on_state_registered()` function returns. Once the request for unread notifications is successfully delivered, `PUSH_SERVICE_ERROR_NONE` is returned."	

Claim 13

13. The message link server of claim 1, wherein one of the message delivery triggers is the receipt of a particular network element message from one of the network elements.	The accused instrumentalities comprise the "message link server of claim 1, wherein one of the message delivery triggers is the receipt of a particular network element message from one of the network elements." As explained *supra* for claim 1, app servers may communicate with push messaging servers to cause messages to be sent to user devices.

Claim 15

15. A method of operating a message link server, comprising:	*See* supra claim 1.
maintaining a respective secure message link through an Internet network between the message link server and a respective device link agent on each of a plurality of	The accused instrumentalities comprise a method including "maintaining a respective secure message link through an Internet network between the message link server and a respective device link agent on each of a plurality of wireless end-user devices, each of the wireless end-

wireless end-user devices, each of the wireless end-user devices comprising multiple software components authorized to receive and process data from secure message link messages received via a device link agent on that device;	user devices comprising multiple software components authorized to receive and process data from secure message link messages received via a device link agent on that device." *See* supra claim 1[a].
receiving network element messages from a plurality of network elements, the received network element messages comprising respective message content and requests for delivery of the respective message content to respective wireless end-user devices, the respective message content including data for, and an identification of, a respective one of the authorized software components;	The accused instrumentalities comprise a method including "receiving network element messages from a plurality of network elements, the received network element messages comprising respective message content and requests for delivery of the respective message content to respective wireless end-user devices, the respective message content including data for, and an identification of, a respective one of the authorized software components." *See* supra claim 1[b].
buffering content from the received network element messages for which delivery is requested to a given one of the wireless end-user devices;	The accused instrumentalities comprise a method including "buffering content from the received network element messages for which delivery is requested to a given one of the wireless end-user devices." *See* supra claim 1[c].
determining when one of a plurality of message delivery triggers for the given one of the wireless end-user devices has occurred, wherein for at least some of the received network element messages, the receipt of such a message is not a message delivery trigger, and for at least one of the message delivery triggers, the trigger is an occurrence of an asynchronous event with time-critical messaging needs; and	The accused instrumentalities comprise a method including "determining when one of a plurality of message delivery triggers for the given one of the wireless end-user devices has occurred, wherein for at least some of the received network element messages, the receipt of such a message is not a message delivery trigger, and for at least one of the message delivery triggers, the trigger is an occurrence of an asynchronous event with time-critical messaging needs." *See* supra claim 1[d].
upon determining that one of the message delivery triggers has occurred, supplying one or more messages comprising the	The accused instrumentalities comprise a method including "upon determining that one of the message delivery triggers has occurred, supplying one or more messages comprising the

buffered content for delivery on the secure message link maintained between the message link server and a device link agent on the given one of the wireless end-user devices.	buffered content for delivery on the secure message link maintained between the message link server and a device link agent on the given one of the wireless end-user devices." *See* supra claim 1[e].